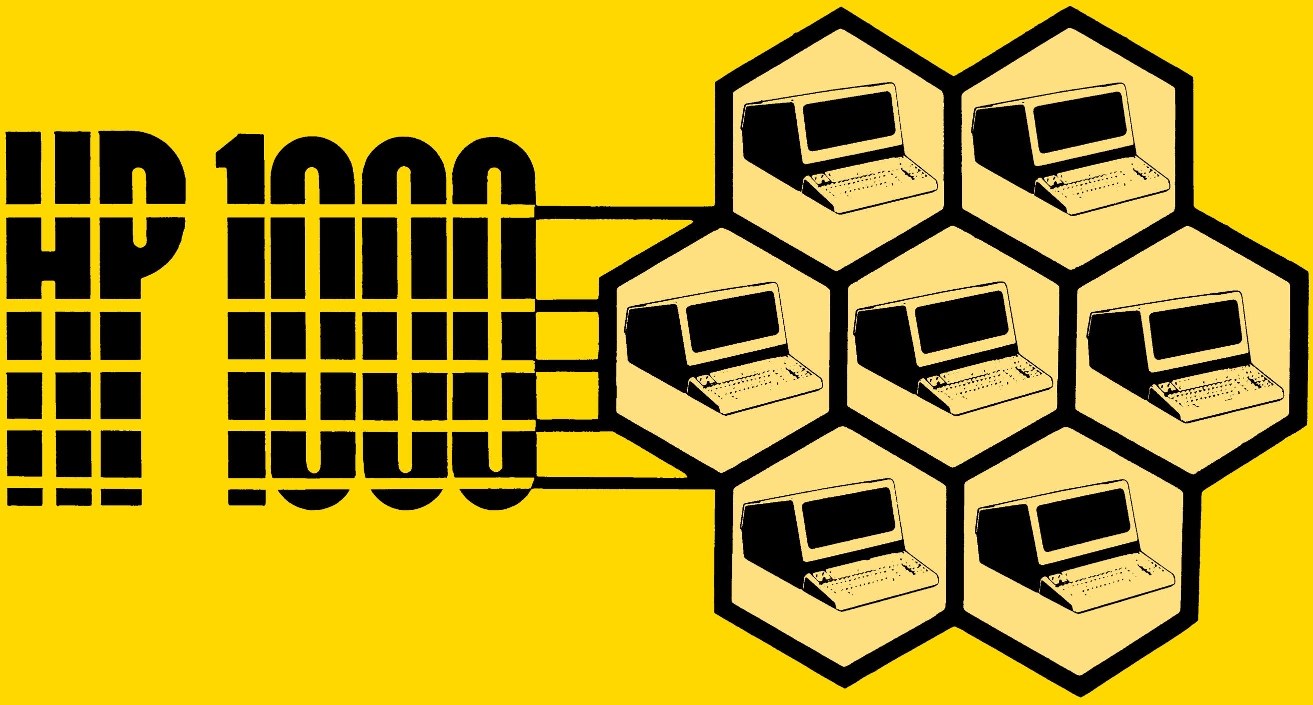
HEWLETT PACKARD

IMAGE/1000
92069A and 92073A Data Base Management Systems

Reference Manual



92069A and 92073A IMAGE/1000 Data Base Management Systems

Reference Manual

HEWLETT PACKARD

HEWLETT-PACKARD COMPANY
Data Systems Division
11000 Wolfe Road
Cupertino, California 95014

Update 2 (July 1981)
MANUAL PART NO. 92069-90001
Printed in U.S.A. July 1980

PRINTING HISTORY

The Printing History below identifies the Edition of this Manual and any Updates that are included. Periodically, Update packages are distributed which contain replacement pages to be merged into the manual, including an updated copy of this Printing History page. Also, the update may contain write-in instructions.

Each reprinting of this manual will incorporate all past Updates, however, no new information will be added. Thus, the reprinted copy will be identical in content to prior printings of the same edition with its user-inserted update information. New editions of this manual will contain new information, as well as all Updates.

To determine what manual edition and update is compatible with your current software revision code, refer to the appropriate Software Numbering Catalog, Software Product Catalog, or Diagnostic Configurator Manual.

Second Edition	Jul 1980
Update 1	Jan 1981
Update 2	Jul 1981

Conventions Used In This Manual

Notation — Description

[]

An element inside brackets is optional. Several elements stacked inside a pair of of brackets means the user may select any one or none of these elements.

```
Example:  A   user may select A or B or neither
          B
```

{}

When several elements are stacked within braces the user must select one of these elements.

```
           A
Example:   B   user must select A or B or C.
           C
```

uppercase

Uppercase letters means that the information should be presented exactly as shown.

lowercase

Lowercase denotes a parameter which must be replaced by a user-supplied variable.

names

All item names, data set names, database names and level code words consist of one to six ASCII characters restricted as follows:

o only printable characters, ! through _.
o not allowed are: plus (+), minus (-), colon (:), comma (,), semicolon (;), left parenthesis ((), right parenthesis ()), period (.), equal sign (=), double quotes ("), underscore (_) or embedded spaces.
o the first character must not be blank (space) or a number.

file names

References to file names conform to RTE-L FMP conventions (with the restrictions listed above). Any file name referred to as an FMP namr consists of a file namr or a logical unit number. Any file name

	number. Any file name referred to as an FMP file namr consists only of a file namr; a logical unit number is not permitted. In an FMP file namr, negative lus are not permitted; only cartridge reference numbers can be used.
log lu	The log lu is LU 1 if the terminal being used is a Session terminal, the terminal lu if the terminal is a Multi-Terminal Monitor terminal and LU 1 if the terminal is neither a Session nor MTM terminal.

Preface

This manual describes the IMAGE/1000 Database Management System. IMAGE is a complete software package for consolidating individual data files into a single, interrelated database. Two IMAGE/1000 products, identified by part numbers 92073A and 92069A, are described in this manual. The 92069A product contains all the features of 92073A IMAGE, as well as a BASIC/IMAGE interface and QUERY. QUERY allows a non-programmer to interact with the database using simple, English-like commands.

The RTE-L operating system supports the 92073A product, while RTE-IVB and RTE-XL support both the 92073A and 92069A products. Since this manual is compatible with both products, 92073A IMAGE users should ignore sections concerning QUERY and BASIC language access.

In addition to being a functional reference manual, this document provides tutorial aid for inexperienced users through a conceptual approach to database management. Chapter 1 discusses what a database is, why it is used, and how to best make use of database concepts through IMAGE/1000 features. The remaining chapters describe the use of IMAGE/1000 software from CREATING THE DATABASE to HOST LANGUAGE ACCESS to MAINTAINING THE DATABASE. QUERYING THE DATABASE (92069A users only) fully describes that interactive utility. Experienced users will appreciate the HOST LANGUAGE EXAMPLES and ADVANCED TOPICS chapters. ADVANCED TOPICS will aid in optimizing database designs through knowledge of system internals.

The appendices contain listings of the HP character set, IMAGE error messages, and a comparison of 92069A IMAGE with the 92073A and 92063A IMAGE products. The error messages of Appendix B will be referred to by all users, while Appendix C will be of greatest interest to those who have 92063A databases and desire to upgrade them to 92069A IMAGE. Appendix D fully describes the differences between the 92069A and 92073A products.

This manual assumes a basic knowledge of RTE Operating Systems and at least one programming language (FORTRAN, PASCAL, BASIC, or Assembly). Familiarity with Distributed Systems/1000 is also helpful if Remote Database Access will be used.

Contents

Illustrations

Tables

Chapter 1
Designing The Database

As computer technology has advanced, methods of controlling data have become more and more sophisticated. Data handling has progressed from simple sequential files through random file access to integrated database management. Database management systems offer increased capabilities to the user, but by the nature of the added features, more planning is necessary for optimal operation. Time invested initially in planning will pay off once the database has been established. This chapter discusses some basic concepts in designing and implementing a database, as well as a general look at the structure and features of IMAGE.

What Is A Database?

A database is a collection of logically related files containing both data and structural information. Pointers within the database allow a user to gain access to related data and to index data across files. The organization of a database may take one of several forms; two examples are a hierarchical structure and a network structure.

The hierarchical structure is a natural growth from earlier conventional file management techniques. Data must be accessed through levels of qualifiers. For example, to get to information about an employee, information about division and department must first be accessed. Cross reference files and linkage files are extensively used to relate files for logical association and accessibility. When the number of data files grows and the interrelationships among them become more complex, the requirement for cross reference files and linkage files tends to increase exponentially. An inherent result is that more overhead is required to access data.

The network database is structured under the premise that only when one logical group of data (data set or file) is related to another logical group of data, is a direct linkage constructed between them. Thus, separate cross reference files and linkage files are no longer required. The increase of complexity is directly proportional to the number of direct relationships existing among a number of logical data sets. When logical data sets are considered as nodes with direct accessing paths connecting them, a network database is formed.

Objectives Of A Database Management System

An effective database organization method essentially involves maintenance of the database. However, there are many implications and practical considerations when one considers a database management system.

Objectives of a database management system are data integrity and security, convenient management, immunity to external changes, and readiness of use in different environments. In order to achieve these objectives, the independence of a data base is most important. Since data is centralized, it becomes natural to centralize data control. This centralization provides for the independence of the database from application programs. Additions and modifications to application programs should then not affect the data base and additions and modifications to the database should not affect existing application programs. The independence of a database provides the foundation for an information system to meet the demand for dynamic changes and growth. The database management system must be user-oriented so that it can be learned quickly, understood easily and used conveniently for a wide spectrum of applications. Facilities should be provided to the user for security, database recovery and debugging aids.

These major points should be the objective of any database management system. IMAGE adopts a network database organization and meets the criteria of an effective database management system.

Why Use A Database?

The primary benefit derived from use of the IMAGE database management system is time savings. These savings are typically manifested in the following areas:

- File Consolidation

  Most information processing systems that service more than one application area contain duplicate data. For example, a vendor's name may appear in an inventory file, an accounts payable file and an address label file. The data stored in these three files probably varies slightly from file to file, resulting not only in wasted file space but also inconsistent program output. Redundant and inconsistent information severely dilutes any system's capacity to deal with large amounts of data.

  File consolidation into a database eliminates most data redundancy. Through the use of pointers, logically related items of information are chained together, even if they are physically separated. In the example of vendor names and addresses, only one set of data would be stored. Through the use of logical associations, the data could be used by any program needing it. Since there is only one record to retrieve and modify, the work required for data maintenance is greatly reduced. Finally, all reports drawn from that item of information are consistent.

- Program File Independence

  Conventional file structures tend to be rigid and inflexible. The nature of conventional file management systems requires that the logic of application programs be intricately interwoven with file design. When it becomes necessary to alter the structure of a file, a program must be written to change the file, and programs that access the file must be changed to reflect the file change. Since change is the rule rather than the exception in data processing, a large percentage of total time and manpower is spent reprogramming.

  IMAGE allows the data structure to be independent of the application program. Data item relationships are independently defined. Changes in the database structure need only be incorporated into those programs that manipulate the changed data. User programs need view only that portion of the database description that pertains to each program's processing requirements. Since all references to the database are resolved at execution time, only those programs affected by changes to the database description need be changed.

- Versatility

  Conventional file organization techniques allow limited access to the data they contain. Most structures allow single key access with additional relational access available only through the implementation of extensive application level programming support.

  IMAGE allows data to be accessed with multiple keys as well as through a variety of other access methods.

- Data Security

  Conventional file management systems contain extremely limited data security provisions. Access to computer readable data may be denied to individuals with system access only by providing physical protection for the media upon which the file is stored; for example, the use of a vault for storage of sensitive data stored on magnetic tape.

IMAGE provides security at the file and data item level. The implementation of security at the item level allows sensitive data to be stored on-line under the control of a database manager or designer, with minimal regard for additional security provisions. IMAGE security provisions can limit even programmer or operator access to extremely sensitive information.

While implementing a new application system IMAGE can be expected to save time in the following ways:

- Program Development

  The database structure can be defined and built without the use of special purpose application level programming. Since control of the linkage portion of the database is under IMAGE software control, the programmer need not be concerned with testing the structure and can concentrate on the functional programming task at hand.

- Program Maintenance

  Throughout the life of a system, processing requirements evolve as the usefulness of the data is explored. As file organization concepts change with the needs of the application, some data restructuring can be done with little impact on existing programs. Changes to the structure of an existing database affects only those programs that process the changed data; no other programs in the system need be recompiled to reflect the new database structure.

  The evolution of the database is not limited by the need to balance the cost of changing an existing systems against the benefits to be derived from the new structure. It is not necessary to do a "where-used" evaluation of a data item carried in multiple files to assess the impact of a data change on existing systems.

In summary, effective use of IMAGE can remove a large portion of the overhead associated with integrated system design from the shoulders of applications analysts and programmers. It affords the opportunity to channel system design talents into functional rather than structurally-supportive design tasks.

Data Set Relationships

The direct relations among different groups of data is the foundation of an IMAGE database. When one examines the various day-to-day applications, abundant examples of one group of data related to another group of data which relate to still another group of data are readily found, such as:

- Open orders and inventory orders
- Finished items and components
- Production orders and labor tickets
- Labor tickets and employees
- Employees and skill classifications
- Parts under test and test results

For the purposes of this manual, we will examine the last instance, in the form of a quality assurance (QA) database. As each assembly is tested, failures are reported, and the information stored includes the particular part that failed, its location in the assembly, and some code to indicate the type of failure.

In the QA example, it is desirable to be able to access information about any assembly quickly. Since there are failures involved with each assembly, the efficient processing of transactions corresponding to respective assemblies is important. A logical approach to organize the two sets of information and their association is:

1. To have the information about each assembly in an entry which is accessible directly by an assembly number. All the assembly entries are grouped together to form a data set or file. Furthermore, such a data set will be considered a master data set because entries in it may be accessed directly and the entries will be called master entries.

2. To have the information about each failure in an entry, and all such entries grouped together to form another type of data set. Failures of a particular assembly will be linked together in a chain which in turn will be linked to the associated assembly entry in the master data set. The failure entries collectively comprise a detail data set. A detail data set is attached to one or more master data sets since a particular entry must be accessed through a master data set. The entries in a detail data set are called detail entries.

Figure 1-1 shows the two related data sets and further shows that:

- Information about a particular assembly can appear in an entry with that assembly number.
- An assembly entry is chained together with a group of failure entries because of its association with those failure entries.
- An assembly entry can be accessed directly by an assembly number, which is called the key.
- From an assembly entry, the first associated failure entry in a chain can be accessed through a linkage path. From the first failure record, the second transaction entry in the chain can be accessed, and so on.

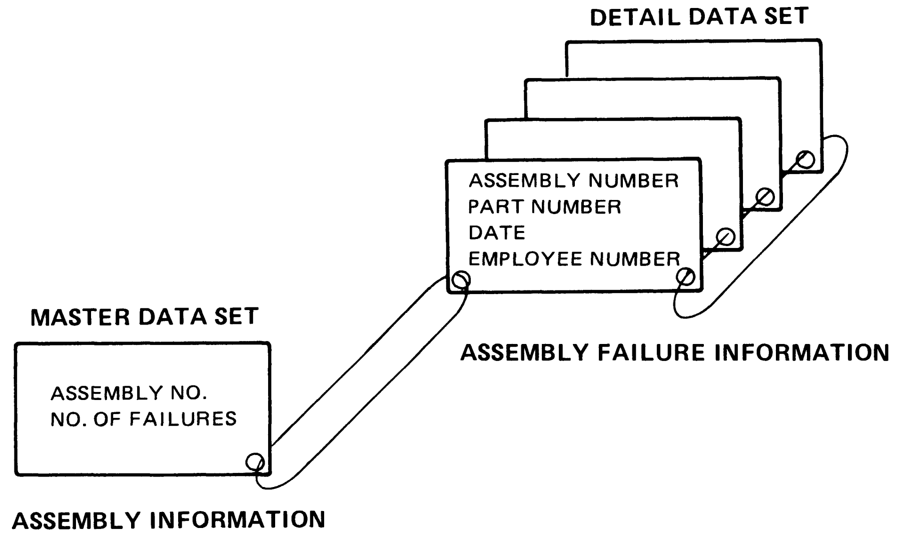


Figure 1-1. Master and Detail Data Set

It is important to note that the linkage path information is stored within a data record instead of elsewhere.

Now the association can be expanded beyond two data sets. Again, in a typical situation, it may be desirable to know what parts constitute a given assembly. If the bill of materials information for each assembly is kept in an entry, another detail data set can be established. Each part for a given assembly will be linked to the corresponding assembly entry in the assembly data set. Now there are two detail data sets associated with one master data set as shown in Figure 1-2.

On the other hand, each assembly failure involves a particular part. If an entry is used to describe each part, a master data set can be established to include all such entries. An association can be made between the failure data set and the part number data set. Thus, failures of the same part will be linked together to form a chain which in turn, will be linked to the

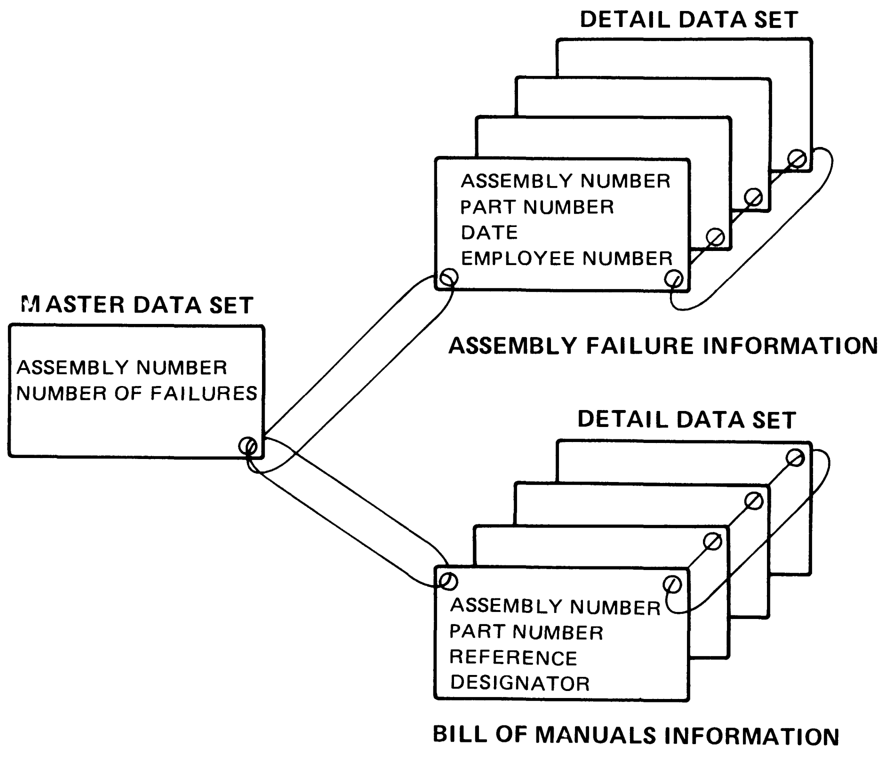


Figure 1-2. Master and Two Detail Data Sets

entry describing that part. In this case, one detail data set associates with two master data sets as shown in Figure 1-3.

By following the same principle, data sets can be added and associations established among them as dictated by needs.

Database Structure

Having been introduced to the structure of the IMAGE database, let us look at an application and design a possible solution to

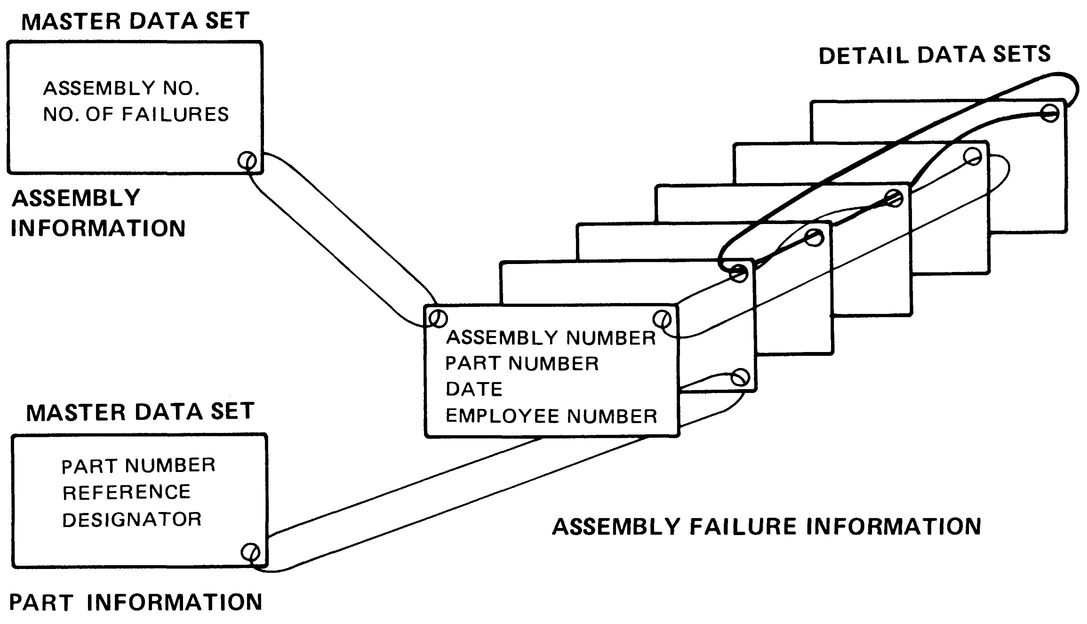


Figure 1-3. Two Masters and One Detail Data Set

the database problems using IMAGE.

An understanding of the database structure is necessary before the database can be designed. The following paragraphs describe the various data elements and their relationships.

Database

An IMAGE database consists of one or more data sets which have some logical relationship to one another. These data sets are stored on disc as FMP files. A data set consists of one or more fixed length data entries (logical records). A data entry consists of one or more data items (fields). Figure 1-4 illustrates the organizational hierarchy of a typical IMAGE database.

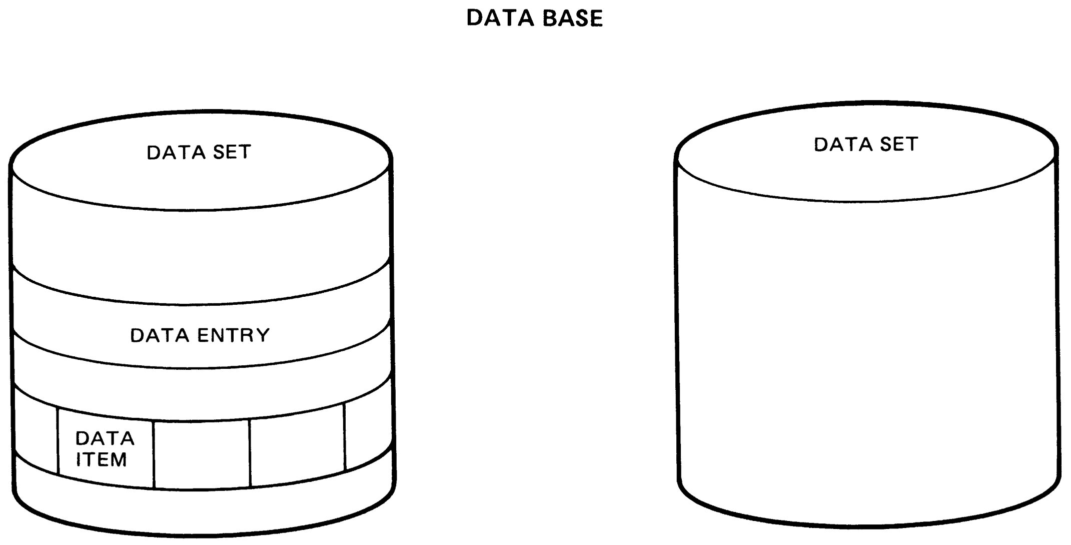


Figure 1-4. IMAGE Database Components

Data Items

A data item is the smallest accessible data element in a database. Each data item consists of a value referenced by a data item name, typically selected to describe the data value. In general, several item values are referenced by the same data item name, each value existing in a different data entry.

Compound Data Items

A compound data item is a named group of identically defined, adjacent items within the same data entry. Each occurrence of the data item is called an element and each element may have a value. A compound item is similar to an array in programming languages such as FORTRAN. A data entry might contain a compound item named DAILY with 5 elements in which the total tests for each day are recorded.

Data Types

The database designer defines each data item as a particular type depending on what kind of information is to be stored in the item. It may be integer, real or ASCII character information. The data types are described in detail in Chapter 2.

Data Entries

A data entry is an ordered set of data items. The order of data items in an entry is specified when the database is defined. Data entries may be defined with at most 127 data item names, none of which is repeated within the entry. The length of the data entry is the sum of the lengths of the data items it contains. When the entry is stored in the database, additional structural information will be added to form the complete record.

Data Sets

A data set is a collection of data entries, called occurrences, where each entry contains values for the same data items. For example, a data set containing occurrences of test failures could have items such as assembly number, part number, date, failure code, failure location, point values and employee number. Each data set is referenced by a unique data set name. Each data set is stored in one disc file consisting of storage locations called records. When the database is described with the database definition language, the capacity or number of records of each data set is specified. Each record is identified by a record number which is used by IMAGE to retrieve the entry within. This record number is called a relative record number. When the capacity of a data set is specified, the file is created with space for that number of entries. If a record at any given relative record number does not have an entry in it, it is said to be an empty record.

Master Data Sets

Master data sets are characterized in the following ways:

- They are used to keep information relating to a uniquely identifiable entity; for example, information describing an assembly, where the key value is that which uniquely identifies the entity.
- They allow for rapid retrieval of a data entry since one of the data items in the entry, called the key item, determines the location of the data entry. A key item may not be a compound item. The assembly data set could contain a key item of assembly number. The location of each entry is determined by the value of the assembly number.
- They can be related to detail data sets containing similar key items and thus serve as indices to the detail data set. The assembly number key item in the assembly master data set could point to an assembly number key item in a failure transaction detail data set.

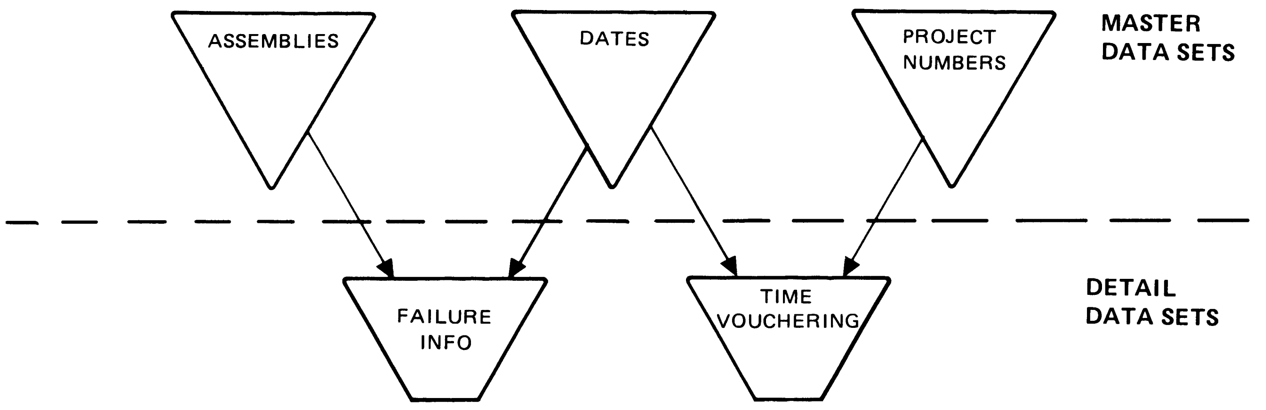
Figure 1-5 illustrates linkage between data sets. Although there may be unused storage locations in the assembly data set, IMAGE disallows any attempt to add another data entry with assembly number 9206018001. The key item value of each entry must remain unique. The values of other data items in the master data set are not necessarily unique. This is because they are not key items and are not used to determine the location of the data entry.

Detail Data Sets

Detail data sets are characterized in the following ways:

- They are used to record information about related events; for example, information about all failures of the same assembly.
- They allow retrieval of all entries pertaining to a uniquely identifiable entity. For example, assembly number 9206018001 can be used to retrieve information about all failures of that assembly.

The storage location for a detail data set entry has no relation to its data content. When a new data entry is added to a detail data set, it is placed in the first available location.



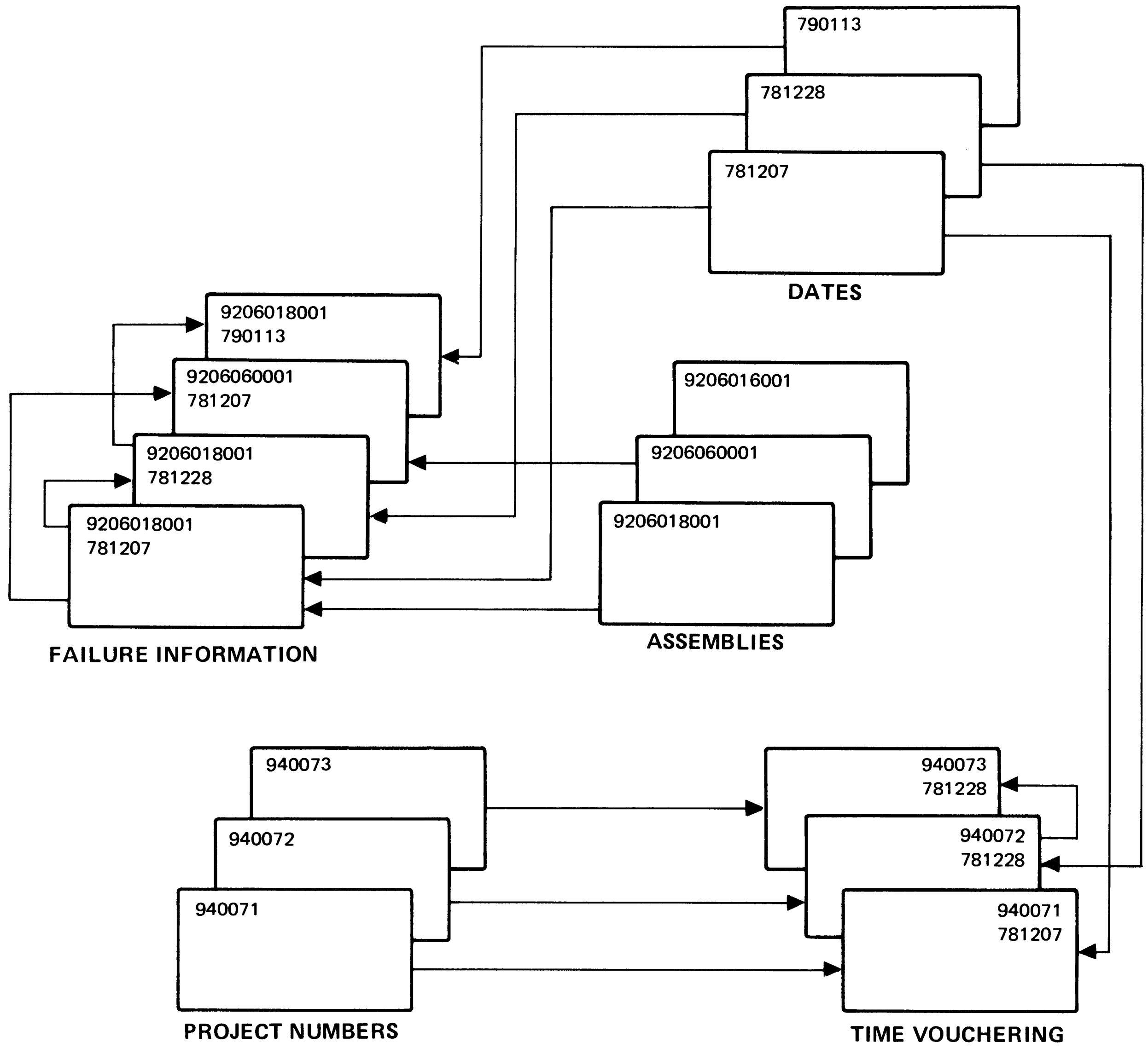


Figure 1-5. Set Relationships

Unlike a master data set which contains at most one key item, a detail data set may be defined with from zero to sixteen key items. The values of a particular key item need not be unique. Generally, a number of entries will contain the same value for a specific key item.

IMAGE stores pointer information with each detail entry which links together all entries with the same key item value. Entries linked together in this way form a chain. A key item is defined for a detail data set if it is desired to access all entries with a common key item value, in other words, all entries in a chain.

Sorted Chains

A CHAIN consists of all detail data set entries accessed through the same key item value. IMAGE contains a SORTED CHAIN feature which places entries in a chain in ascending order, sorted by the value of any other item in the entry.

For example, referring to Figure 1-7 (Database Structure), the detail data set TIME has a key item PROJECT NUMBER. Entries sharing the same PROJECT NUMBER could be sorted according to any of the remaining items in the data set: EMPLOYEE NUMBER, DATE, or HOURS. For the Accounting Department it would be most useful to view a particular project's time vouchering data in the order in which the work was done - that is, sorted by DATE. For this reason, PROJECT NUMBER will be sorted by DATE in the detail data set TIME.

When using the sorted chains feature, IMAGE will take slightly longer to place an entry in a chain. This is because the entry's position must be determined by searching the existing chain. The amount of search time is dependent on the number of entries in the chain. For this reason, sorted chains should be used only where large amounts of sorting must be done on a regular basis. In such a case, the extra time needed to insert each data entry in sorted order will be justified by the time saved when the data is retrieved.

If the SORTED CHAINS feature is not used, entries will be placed in the chain in the order that they were entered. This chronological ordering is the default of the SORTED CHAINS feature.

Paths

A master data set may have only one key item and it may be related to one or more detail data sets. The key items in master and detail sets must be of the same type and size. This relationship forms a path. A master set may have paths to up to sixteen different data sets. In Figure 1-5, the assembly number key item in the assembly data set and the assembly number key item in the failure information data set link [illegible] the two data sets together, forming a path.

[illegible]

A detail data set can have up to sixteen key items, linking it to up to sixteen different master data sets. In Figure 1-5, the failure data set is linked to the assembly data set and the date data set. A detail set may have two or more links to the same master data set.

For each path from a master data set, there is a chain for each key item value. This chain consists of all detail set entries whose key item along a path equals the related master set's key item value. The master entry contains pointer information to the chain. This information is called a chain head. The format of the chain head is shown in Chapter 7. For example, the date master entries shown in Figure 1-5 contain two chain heads, one for failure entries and one for time vouchering entries. Chain heads are maintained automatically by IMAGE.

Within a detail set, all entries that share a key item value are chained together. Each detail entry will belong to as many chains as there are key item values in the set. A chain can contain zero entries or as many as there are entries in the data set. In Figure 1-5, in the failure data set, there are three entries in the chain for assembly number 9206018001 and one entry for assembly number 9206060001.

Automatic and Manual Masters

A master data set may be automatic or manual. These two types of masters have the following characteristics:

MANUAL	AUTOMATIC
May be stand-alone. Need not be related to any detail data set.	Must be related to one or more detail data sets.
May contain data items in addition to the key item.	Must contain only one data item, the key item.
Entries must explicitly be added or deleted. A related detail data entry cannot be added until a master entry with matching key item value has been added. When the last detail entry related to a master entry is deleted, the master entry still remains in the data set. Before a master entry can be deleted, all related detail entries must be deleted.	IMAGE automatically adds or deletes entries when needed based on the addition or deletion of related detail data set entries. When a detail entry is added with a key item value different from all current key item values, a master entry with matching key value is automatically added. Deletions of detail entries trigger an automatic deletion of the matching master entry if it is determined that all related data chains are empty.
The key item values of existing master entries serve as a table of legitimate key item values for all related detail data sets.	

In Figure 1-6, PROJ is a manual master data set and DATEF is an automatic master. Before the TIME entry for project number 940072 is added to TIME, PROJ must contain an entry with the same project number. However, a DATEF entry for DATE equal to 780105 is automatically added by IMAGE when the detail entry is added to TIME, unless it is already in the DATEF data set.

Note that DATEF contains only one data item, the key item DATE, while PROJ, which is a manual master, contains several data items in addition to the key item.

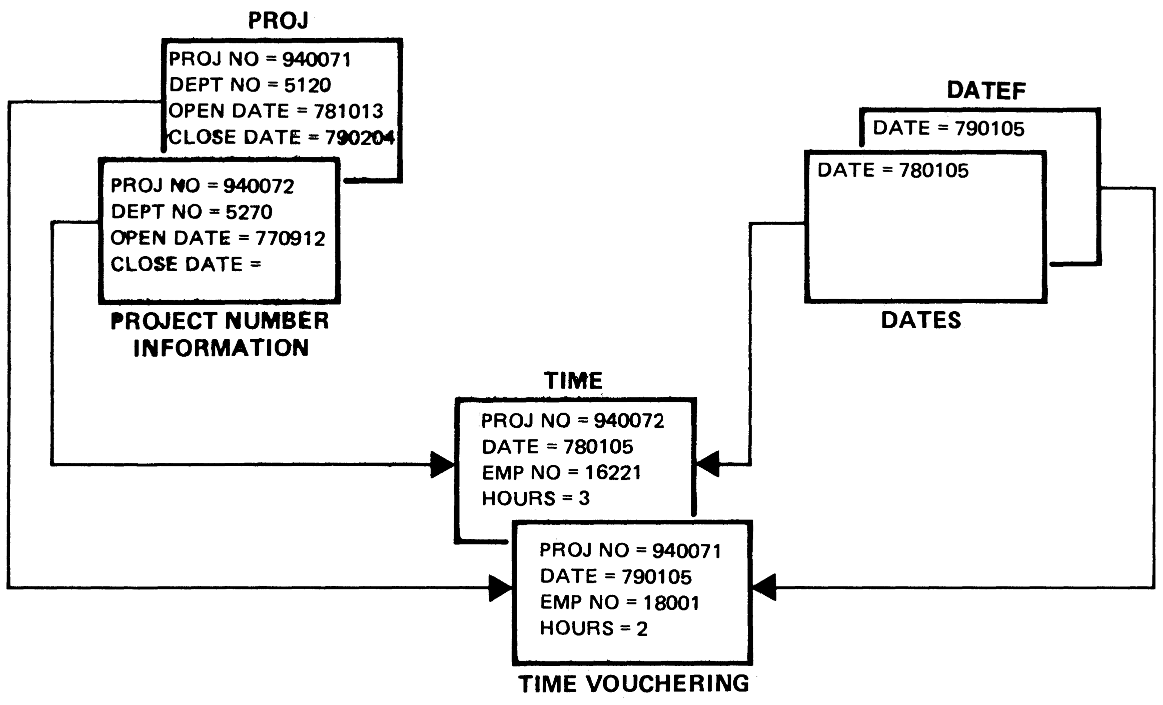


Figure 1-6. Master and Detail Data Set Interaction

If the TIME entry with project number 940071 is deleted and no other TIME entry contains a DATE value of 790105, the DATEF entry with that value is deleted automatically by IMAGE.

Manual vs. Automatic Masters

Database designers may use:

- manual masters to ensure that valid key item values are entered for related detail entries, or
- automatic masters to save time when the key item values are unpredictable or so numerous that manual addition and deletion of master entries is undesirable.

Whenever a single data item is sufficient for a master data set, the database designer must decide between the control of data entry available through manual masters and the time-savings offered by automatic masters. In the QA Database (Figure 1-7), DATEF is an automatic data set, but PART, which also has one item, is a manual set, so that control over part numbers entered into the database can be maintained.

A manual master may be defined in the data schema as having zero data paths. With such a definition, a manual master serves as a special type of detail data set:

1. one which is randomly organized

2. one which is not linked to any master data sets

item

4. one whose data entries can be quickly accessed by key value via the hashing algorithm

Database Files

All database elements are stored in RTE FMP files. The files are created when the database designer executes the Schema Processor (DBDS). The cartridge or cartridges on which the database files reside are specified in the schema that is the input to the Schema Processor. The security code of all database files will be equal to that security code specified as the security code of the database. Note that although the security code is specified in the schema as a

Data Files

There is one data file for each data set of a database. At creation time, the size of each record and number of records in the file are determined by information in the root file. The data files are created and initialized by DBDS at the same time the root file is created.

Each data file has the same security code as the root file. The name of each data file is equal to the name of the data set. A data file may or may not reside on the same cartridge as the root file.

The size of a data file is determined by the characteristics of the data set it contains, which include the size of the entries, the number of paths to or from the set and the capacity. When the file is created, it is created to account for the total capacity, regardless of the number of entries the set actually contains. Records which have no entries assigned to them are called empty records.

Developing A Database System

In developing a viable database system, the following questions should be asked:

1. Who will use the database?
   Many times the database is designed and maintained by one group to make data available to many other groups. The needs of all of the users of the database should be considered before the database is designed to insure that these needs can all be met in the most efficient manner. Having to retrofit features and capabilities into a database after it has been created will be far less efficient than having allowed for all users in the beginning.

2. What application programs need to be written?
   Before an application program is written, the programmer needs to define the input and output to that program. Application programs interfacing to a database have at least part of their input and output needs met by the database. Defining the application programs to be written and their input and output needs will allow the database to have the capability to service those needs.

3. What will the data items and data entries look like?
   At this point, all of the database users need to define their data items and data entries and these definitions should be collected and examined for redundancy and completeness. Conflicting needs of different users should be resolved at this time.

4. What will the key items be?
   Having defined the application programs and the data items, the key items can now be identified. As many key items as are necessary can be defined so that all users may able to access the database in the most efficient manner for them.

5. What are the set relationships?
   The set relationships will be determined by the key items needed, the input and output requirements of the application programs and the contents of the data entries.

6. What are the back-up requirements?
   The back-up requirements of the database will vary according to usage. If the database is referenced frequently, but data is entered only periodically, the database may not need to be backed-up daily. However, if the database is constantly being updated, an intermediate back-up should be performed at least once a day, with a more complete back-up less frequently. It must also be determined if the standard back-up utilities that are available are sufficient or if some unique need warrants special application programs to provide additional security.

Let us use these six steps to design a database to be used in the Quality Assurance Department of a large manufacturing company. XYZ Manufacturing produces various products, all using electronic assemblies as part of their components. XYZ Company produces the electronic assemblies at their plant. When an assembly fails under test, it is sent to Product Assurance for analysis and repair. The Product Assurance people are having difficulty keeping track of failure information, so they propose that a database be set up to aid them in this area.

Once the decision is made to implement a database management system, the first question must be asked. Who will use the database? The main users will be the Product Assurance group, who would like to be able to access failure information in a number of ways. In addition, the Accounting Department is interested in using the same database to accumulate time vouchering information. Manufacturing also believes the information will be of use to them to get some idea of the number and types of failures of different parts.

Having determined the users, the planning group began to examine what programs were to be written and also what kinds of reports were to be generated. The program of the highest priority was one which prompted the technician for failure information and also updated a count of the number of failures for a given assembly. As time went on, the program could be enhanced to add additional checks on the information entered.

The Manufacturing people wanted a program which would instantly give them failure information about a given part or assembly. The Accounting Department needed the ability to add, delete or close project numbers and a program to enter time vouchering information that would check for an existing but inactive project number.

The Product Assurance people plan reports that will list date, assembly number, part number, failure code and reference code. They would like to access failure information through any one of the following items: assembly number, part number, date, reference code, failure code, employee number, or supervisor number.

Each group listed the items they felt they needed in the database. There were some duplications. Product Assurance wanted assembly number, part number, date, failure code, reference designator, employee number, supervisor number, and an array of test values. Accounting wanted project number, department number, open date, close date, employee number, date and hours worked. Manufacturing wanted assembly number, part number and cumulative count of failures for a given assembly. Based on the requirements for accessing and reporting, the following items were identified as key items: assembly number, date, part number, failure code, reference designator, employee number and project number.

Having defined the key items and the non-key items, the planning group began to build the set relationships. Since Product Assurance needed access to the failure data set via a number of different key items, five master/detail links were created. The master sets they needed were for assembly number, part number, date, reference designator and failure code. The assembly number set needed to contain additional information besides just the assembly number, so it had to be a manual master. It was thought to be desirable to control the entering of part numbers through a master data set, so the part data set was also made a manual master set. The other three master data sets, failure code, reference designator and date were made automatic master data sets.

Accounting wanted to key into the failure information and the time vouchering data set by employee number, so that was made an automatic master. They also needed a project number data set with additional information, so that data set was made a manual master.

The design of the database that will fit all of the above requirements is shown in Figure 1-7. After designing the database, the planning group discussed back-up requirements. Based on the daily number of transactions anticipated, it was felt that a daily back-up would be necessary. Also, once a week, more extensive maintenance would be done that would include both back-up and verifying database integrity. In addition, the planning group left open the possibility of altering the failure transaction program at some time to include the logging of each transaction as it is entered for additional back-up.

The database has been designed to meet today's requirements and hopefully to allow for graceful modification and expansion in the future.

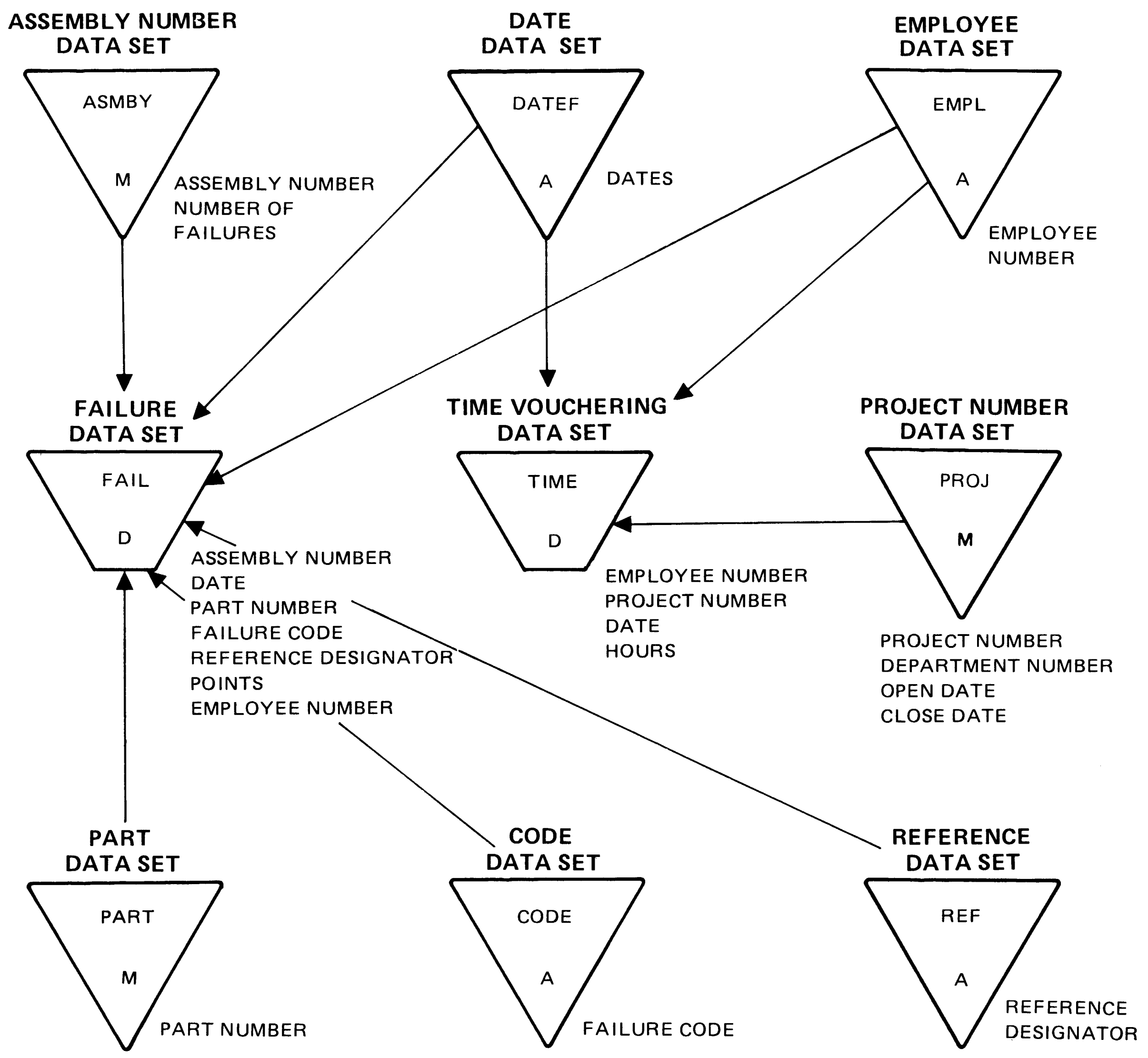


Figure 1-7. QA Database Structure

Chapter 2
Creating The Database

Once the database has been designed, it must be described with the database description statements and processed by the Schema Process (DBDS) to create the root file. Figure 2-1 illustrates the steps in defining the database.

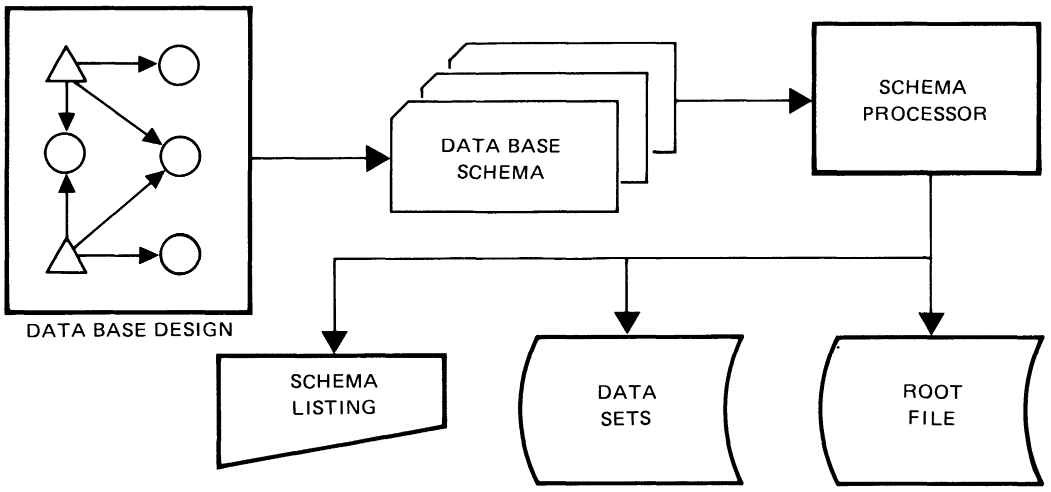


Figure 2-1. Database Design Process

DBDS processes a description of the user database (called a schema) and produces an internal system description of the database (called a root file) and all data sets used by the IMAGE system. The root file describes the relationship between data items, the level of security associated with each item and the relationships between master and detail data sets in the database. The root file exists as a RTE disc file.

Using DBDS consists of four main steps as shown in Figure 2-1:

1. Design of the database by the database manager.

2. Describing the design (creating a schema).

3. Processing the schema by DBDS.

4. Creation of the root file and all data sets by DBDS.

Language Conventions

The database description, called a schema, may exist in the RTE system as a disc file. Regardless of the actual physical record size of the file, DBDS reads, prints, and processes only the first 72 characters of each record. Any remaining character positions in the record are available for comments or collating information.

Comments take the form: <<comment>>. They may contain any characters and may appear anywhere in the schema except embedded in another comment. They are included in the schema listing but are otherwise ignored by the DBDS processor program. Spaces may not appear between the two comment indicator characters on either side of the comment.

For information about restrictions on item names, set names and code words, see CONVENTIONS USED IN THIS MANUAL.

Database Schema Structure

The actual content of a database schema varies for each database depending upon the needs of the database designer. The structure of the database schema does not vary. The database schema structure is:

```
BEGIN DATABASE: database name;
LEVELS:[level part]
ITEMS: item part
SETS: set part
END.
```

where database name is the FMP file namr of the database root file. It consists of the file name, security code and cartridge reference number.

The level part, item part and set part are described on the following pages. Figure 2-3 contains a complete schema for the QA Database that is used in the examples for this manual.

Defining Levels

The level part of the schema is used to define privacy levels. If the level part is omitted from the schema, then no privacy levels exist for specific data items in the database. Each privacy level is defined by associating one level number with one level code word. The level number is an integer from 1 through 15. When defining items in the item part of the schema, a level number may be used to restrict read or write access to the item. Each privacy level definition is terminated by a semicolon; level numbers and level code words are separated from each other by one or more blanks.

If the level part is omitted, LEVELS:; must still be specified.

The form of the level part is:

```
     LEVELS:
          level-number level-code-word;
          level-number level-code-word;
                       .
                       .
                       .
          level-number level-code-word;
```

During creation of the level part of the database, the database designer specifies a series of privacy level numbers and associated code words. The level numbers are integers from 1 through 15. When a user first enters an IMAGE utility or before the data base can be accessed with program calls, a privacy level code word must be entered which corresponds to the highest level number the user is allowed to access. The user cannot read or modify a data item unless his privacy level is equal to or greater than the level number specified for the data item in the item part of the schema.

Below is an example of the level part of a schema.

```
     LEVELS:
         1    OPER;
         5    ACCT;
        10    ADMIN;
        15    SECUR;
```

Defining Items

The item part of the schema defines the name, size, type, number of elements and privacy levels (one for reading and one for writing) for each data item in a database. Up to 255 data items can be defined in the item part. The definition sequence for each data item takes the following form:

```
     item name,[element count] type[(read level,write level)];
```

where:

item name	is the name of the data item. Each item name in the item part must be unique to the database.
element count	defines the number of elements in a compound data item. An element count is an integer from 1 through 255. When an element count is not specified, a value of one is used indicating a simple item. If the element count is greater than one, the item is called a compound item.
type	defines the amount of space in memory that the data item is to occupy (i.e., the data item type). Three data types are allowed:

I1	Denotes an integer number occupying one word in memory. The value may be any integer in the range of -32768 to +32767.
R2	Denotes a real number occupying two words in memory. The value may be any number in the range of ±1.70 * 10**38 to ±1.47 * 10**-39. Real numbers should not be used as key items because of their round-off characteristics.
Xinteger	Denotes an ASCII character string. The integer following X (with no intervening blanks) denotes the number of characters in the string and must not exceed 255. The number of characters times the number of elements must be an even number.

(read level, write level)	An ordered pair of integers specifying the privacy level needed to read the data item and the privacy level needed to modify the data item. If read and write levels are omitted, DBDS assigns a read level of 0 and a write level of 15 to the data item. The read

level must be less than or equal to the write level which must be from 1 through 15. A level number cannot be specified that was not previously defined in the level part.

The total length of a compound item is the product of the element length times the number of elements in the item. If the compound item is a character type, the length of the element may be an odd number of characters. However, the total length of the item in characters must be an even number.

An example of an item part of a schema is:

```
ITEMS:
     ASSY#,   X10(1,10); <<ASSEMBLY NUMBER FOR ASSEMBLY MASTER>>
                         <<AND FAILURE FILE.                  >>
     FAIL#,   I1(1,10);  <<NUMBER OF FAILURES - ASSEMBLY      >>
     PART#,   X8(1,10);  <<PART NUMBER FOR PART NUMBER MASTER >>
                         <<AND FAILURE FILE                   >>
     FAILCD,  X4(1,10);  <<FAIL CODE                          >>
     REFDES,  X4(1,10);  <<REFERENCE DESIGNATOR               >>
     DATE,    X6(1,1);   <<DATE FOR FAILURE FILE, DATE MASTER >>
                         <<AND PRODUCT TESTED FILE            >>
     EMP#,    X6(1,10);  <<EMPLOYEE NUMBER FOR EMPLOYEE MASTER>>
                         <<FAILURE FILE AND TIME VOUCHER FILE >>
     PROJ#,   X6(1,5);   <<PROJECT NUMBER FOR PROJECT NUMBER  >>
                         <<MASTER AND TIME VOUCHER FILE       >>
     DEPT#,   X4(1,5);   <<DEPARTMENT NUMBER                  >>
     OPDAT,   X6(1,5);   <<OPEN DATE FOR PROJECT NUMBER       >>
     CLDAT,   X6(1,5);   <<CLOSE DATE FOR PROJECT NUMBER      >>
     PNTS,8   X1(1,1);   <<FAIL/NO FAIL ARRAY FOR EACH PIN    >>
     HOURS,   R2(1,1);   <<HOURS PER PROJECT NUMBER           >>
```

Defining Sets

The set part of the schema names the various data sets within the database, indicates which items listed in the item part of the schema belong to which set and links the master data sets to the detail data sets by describing which of the data items are key items. The data schema can define at most fifty data sets. No data entry (data record plus media record) can contain more than 4096 bytes.

Master Data Sets

The form of the set part for defining master data sets is:

```
    NAME:  setname,type;
    ENTRY:
           item name [(path count)],
           item name [(path count)],
           .
           .
           .
           item name [(path count)];
    CAPACITY:  capacity;
```

where:

setname	is the FMP file namr of the data set. It consists of the file name and cartridge reference number. If the security code is specified, a warning message will be issued and the security code will be ignored. All data sets have the security code specified in the database FMP file namr, which is the security code of the root file.
type	is either MANUAL (M) or AUTOMATIC (A).
item name	is the name of a data item belonging to the master data set. At least one item must be defined in each data set. The name must have previously appeared in the item part of the schema. One, and exactly one, item name must be followed by a path count to identify the item as a key item link to a detail data set.
path count	is an integer from 0 through 16 which indicates that the data item is a key item. The number itself indicates how many paths exist from the master data set to detail data sets through the key item link.
capacity	is an integer indicating the number of entries allowed in the data set. On RTE-IVB systems, capacity may be as great as (2**31)-1. On RTE-L and RTE-XL systems, capacity may be as great as 32,767.

Item names in the list under ENTRY: are stored by IMAGE in the order they are listed and must appear only once in the set part for any given data set. However, the same item name may appear in more than one data set description. An automatic master data set must contain only one item, the key item, which has a non-zero path count.

An item which is identified as a key item may have the same name in the master data set and the linked detail data set. If the names are different in master and detail data sets, the item type and length must be the same. Key items must be simple items, and cannot be a compound item. An automatic master key item must have a write level less than or equal to the write level of the linked detail data set key item.

The path count performs two functions. The first is to identify the item as a key item and the second to indicate how many detail data set paths are linked to this master. A path count of two, for example, would indicate that two separate detail data sets are linked to this master. A path count of zero is allowed for manual master sets only and indicates that the master data set is used solely for information and is not linked to any detail set. A key item in an automatic master data set cannot have a path count of zero.

Generally, a master data set should have capacity equal to a prime number or to the product of two or three prime numbers. This yields a more uniform spread of master entries and may increase the speed and efficiency of access to the data. For more explanation of how capacity can affect performance, see Chapter 7.

The following is an example of SETS: entries for manual and automatic master data sets.

```
     NAME:  CODE::43,A;
     ENTRY: FAILCD(1);
     CAPACITY: 997;

     NAME:  PROJ::43,M;
     ENTRY: PROJ#(1),
            DEPT#,
            OPDAT,
            CLDAT;
     CAPACITY: 101;
```

Detail Data Sets

Detail sets are defined in the schema in almost the same form as master data sets.

```
   NAME:    setname,  D[ETAIL]

   ENTRY:   item name [(link[(sort item)])],
            item name [(link[(sort item)])],
            .
            .
            .
            item name [(link[(sort item)])],
  CAPACITY:capacity;
```

where:

setname	is the FMP file namr of the data set. It consists of the file name and the cartridge reference number. If the security code is specified, a warning message will be issued and the security code will be ignored. All data sets will have the security code specified in the database FMP file namr, which is the security code of the root file.
item name	is the name of a simple or compound data item belonging to the detail data set. The name must have previously appeared in the item part of the schema. If the item is a key item, it must be a simple item. At least one item must be defined for each data set.
link	is the set name of a master data set. When a link follows the item name in a detail data set, this indicates that the data item is is a key item. Up to 16 such key items may be defined for each detail data set. A detail set may have one or more links to the same master set.
sort item	is an integer, real, or character string item in the detail data set whose value is used to order entries in the chain. Sort items must be simple items (element count equal to 1). A chain cannot be sorted on its own key item, but may be sorted on any other item defined in the detail data set.
capacity	is an integer indicating the number of entries allocated to the data set. On RTE-IVB systems, capacity may be as great as (2**31)-1. On RTE-L and RTE-XL systems, capacity may be as great as 32,767.

Item names in the list under ENTRY: are stored by IMAGE in the order they are listed, and must appear only once in the set part for any given data set. However, the same item name may appear in more than one set description. A data item used as a key item may have the same name in the detail and the linked master data set. If the names are different in detail and master data sets, the item types and lengths must be the same. There must be as many detail data sets linked to a master as was described in the master's path count.

The link parameter indicates which master data set is linked to the detail set. All links following item names must be those of previously defined masters. For this reason, it is useful to define all master data sets before detail data sets in the set part of the schema.

The multiple linking feature allows a master data set to be linked to a detail set by more than one key item in the detail set. Additional links are defined in the same way as the first (see DETAIL DATA SETS, link parameter) and have the same restrictions. In the following example, detail entries EMP# and SUP# are multiply linked to the master set EMPL.

```
     NAME:  FAIL::43,D;
    ENTRY:  ASSY#(ASMBY),
            DATE(DATEF),
            PART#(PART),
            FAILCD(CODE),
            REFDES(REF),
            PNTS,
            EMP#(EMPL),
            SUP#(EMPL);
 CAPACITY:  1009;
```

If none of the item names in the list have a link following, the detail data set is not linked to any master set, but is used for information only. A stand-alone detail data set can be accessed only with a directed or serial read. Generally, it is preferable to define an information-only data set as a manual master since more efficient access is available through keyed reads.

Sorted entries are placed in the chain in ascending order of the sort item value. A chain cannot be sorted on its own key item, but may be sorted on any other item defined in the detail data set. If duplicate sort item values exist, the new entry is placed in the chain immediately following the last entry with the same sort item value.

In the following example, the detail set time linked to the master set PROJ will have key item PROJ# sorted by DATE.

```
     NAME:  TIME::43,D;
    ENTRY:  EMPL#(EMPL),
            PROJ#(PROJ(DATE)),
            DATE(DATEF),
            HOURS;
 CAPACITY:  1009;
```

Specifying Control Options

The schema CONTROL command is an optional command used by the data base designer to specify control options during processing of the database schema by DBDS. The command is entered as the first record of the schema and only one $CONTROL record is allowed. Its form is:

```
$CONTROL:[parameter,parameter,...,parameter];
```

where parameter is any of the following:

LIST	List each source record on the list device.
NOLIST	Suppress the LIST option. When an error occurs, the offending source record is printed along with an error message.
ERRORS=nnn	Set the maximum numbers of errors allowed to nnn (0<nnn<999). If this number is reached during processing, the schema processor terminates, after printing the message: MAX ERRORS - SCHEMA PROCESSING TERMINATED
ROOT	Inform the schema processor to create a root file if no errors are detected in the schema.
NOROOT	Prevent the schema processor from creating a root file or any data sets. The schema is merely checked for errors.
SET	Create the data sets if no errors are detected in the schema.
NOSET	Suppress creation of the data sets.
FIELD	The schema processor prints a table for each data set describing the start and end of each data item if no errors are detected in the schema.

Heading	Definition
SET NAME	name of the data set.
SET NO.	number of the data set.
ITEM NAME	item name.
ITEM NO.	item number.
ITEM TYPE	defines the item as integer (I), real (R) or character (X).
START WORD	word offset into the data record on which the value of the data item starts.
END WORD	word offset into the data record on which the value of the data item ends.
PATH?	defines whether an item is a path item by entering "YES".
SORT ITEM	name of the SORT ITEM if the item under ITEM NAME is a key item (PATH? is yes) and the path is sorted.

TABLE

The schema processor prints a table of summary information (if no errors were detected) about the data sets after the schema has been scanned by DBDS. The data set information printed under the following heading is:

Heading	Definition
DATA SETNAME	name of the data set
TYPE	A = automatic, D = detail, M = manual
# ITEMS	number of data items in a data set entry.
# PATHS	number of paths defined for the data set.
DATA	combined length (in words) of all data items in the data entry.

	MEDIA	media record length. The length of the media record for each entry in the data set equals 5 + (6*path count) for master sets and 3 + (4*path count) for detail data sets.
	CAPAC	capacity or the number of entries allowed for each data set as defined in the schema.
	CARTRIDGE	disc cartridge reference number on which the data set resides.
NOTABLE	Suppress the printing of the summary table.	

Some examples of $CONTROL commands are:

```
$CONTROL: NOROOT,TABLE;
$CONTROL: ERRORS=2;
```

The default conditions for a $CONTROL command are:

```
LIST
ERROR=100
ROOT
SET
NOTABLE
```

If no $CONTROL record is present, the default conditions will be used. If two or more conflicting options are present, the last named will be in effect.

Running DBDS

DBDS can be run to process a database schema and produce a root file and data sets by first preparing an ASCII file containing the database schema definition.

Start DBDS execution by entering the directive

```
:RU,DBDS,input,list,option
```

where:

input is the FMP file namr of the file containing the database schema definition. The default value is the scheduling lu.

list	is the FMP namr to which the listing is to be written. Default is LU 6.
option	is the characters PU if existing data sets are to be purged before new ones are created.

DBDS reads and processes the schema according to the control options specified in the $CONTROL record. If no $CONTROL record exists, DBDS operates under the default conditions specified above.

It is possible to postpone the creation of a root file until the schema has been processed and is error free. If the $CONTROL command includes the parameter NOROOT, the schema is processed but no root file or data sets are created.

If the ROOT and SET options were specified in the $CONTROL record or if DBDS is operating under the default conditions, the data sets and root file are created after DBDS has processed the database schema. DBDS prints the message

```
     ROOT FILE AND DATA SET FILES CREATED
```

on the list device upon the successful creation of the data sets and root file. However, if the disc is filled before the files are all created, the following message will appear:

```
     ROOT FILE AND DATA SET FILES NOT CREATED
```

When this occurs any data sets created will remain on the disc and must be purged before reexecuting DBDS unless the PURGE option is to be used. When purging an IMAGE data set from the File Manager, the FMP file security code is the negative of the security code specified in the DBDS schema.

If errors occur during schema processing, DBDS prints error messages on the list device. See Appendix B for a description of the error messages printed. Then correct the schema and rerun DBDS. Note that the first occurrence of an error may generate other errors, but that DBDS will recover if possible.

DBDS Messages

DBDS prints two kinds of messages during DBDS execution - error messages (which occur when DBDS discovers an incorrect schema statement) and general messages (which DBDS prints after the schema is processed).

DBDS Error Messages

Whenever DBDS encounters an error in a schema statement it prints the line in error with an arrow pointing to the offending word, followed by the appropriate error message as listed in Appendix B. If the $CONTROL LIST option is active, DBDS prints the message following the offending statement as part of the schema listing. If the NOLIST option is active, DBDS prints the offending statement followed by the error message. It then attempts to recover from the error and continue processing statements.

If a statement terminator such as a semi-colon or a list separator such as a comma is missing, DBDS may ignore an entire schema statement, causing subsequent error messages to occur. For example, Figure 2-2 shows a schema with a semi-colon missing after the REFDES, X4(1,10) statement. Omission of the semi-colon causes DBDS to ignore the following characters up to the next semi-colon which means that DBDS completely ignores the next statement. This causes subsequent errors to occur.

```
      HEWLETT-PACKARD IMAGE/1000 DATABASE DEFINITION PROCESSOR

 $CONTROL:;
 BEGIN DATABASE: QA:100:43;
 LEVELS:
        1    OPER;
        5    ACCT;
       10    ADMIN;
       15    SECUR;
 ITEMS:
       ASSY#,  X10(1,10); <<ASSEMBLY NUMBER FOR ASSEMBLY MASTER>>
                          <<AND FAILURE FILE.                 >>
       FAIL#,  I1(1,10);  <<NUMBER OF FAILURES - ASSEMBLY     >>
       PART#,  X8(1,10);  <<PART NUMBER FOR PART NUMBER MASTER >>
                          <<AND FAILURE FILE                  >>
       FAILCD, X4(1,10);  <<FAIL CODE                         >>
       REFDES, X4(1,10)   <<REFERENCE DESIGNATOR              >>
       DATE,   X6(1,1);   <<DATE FOR FAILURE FILE, DATE MASTER >>
           ^
BAD TERMINATOR - ';' EXPECTED.
                          <<AND PRODUCT TESTED FILE           >>
       EMP#,   X6(1,10);  <<EMPLOYEE NUMBER FOR EMPLOYEE MASTER>>
                          <<FAILURE FILE AND TIME VOUCHER FILE >>
       SUP#,   X6(1,10);  <<SUPERVISOR NUMBER FOR FAILURE FILE >>
       PROJ#,  X6(1,5);   <<PROJECT NUMBER FOR PROJECT NUMBER  >>
                          <<MASTER AND TIME VOUCHER FILE      >>
       DEPT#,  X4(1,5);   <<DEPARTMENT NUMBER                 >>
       OPDAT,  X6(1,5);   <<OPEN DATE FOR PROJECT NUMBER      >>
       CLDAT,  X6(1,5);   <<CLOSE DATE FOR PROJECT NUMBER     >>
       PNTS,8  X1(1,1);   <<FAIL/NO FAIL ARRAY FOR EACH PIN   >>
       HOURS,  R2(1,1);   <<HOURS PER PROJECT NUMBER          >>
SETS:
<<                                                            >>
<<               ASSEMBLY FILE                                >>
<<                                                            >>
NAME:  ASMBY::43,M;       <<ASSEMBLY FILE CONTAINS THE ASSEMBLY>>
                          <<NUMBER AND TOTAL NUMBER OF FAILURES>>
                          <<FOR THAT ASSEMBLY                 >>
ENTRY: ASSY#(1),
       FAIL#;
CAPACITY:409;
<<                                                            >>
<<               PART NUMBER FILE                             >>
<<                                                            >>
NAME:  PART::43,M;        <<PART FILE CONTAINS ONLY PART NUMBER>>
                          <<NOTE THAT ALTHOUGH IT CONTAINS ONLY>>
                          <<ONE ITEM, IT IS A MANUAL MASTER SO >>
                          <<ENTRIES CAN BE CONTROLLED.        >>
ENTRY: PART#(1);
CAPACITY: 4001;
<<                                                            >>
<<                 FAIL CODE FILE                                 >>
<<                                                                >>
NAME:  CODE::43,A;         <<FAIL CODE FILE CONTAINS ALL FAILURE>>
                           <<CODES                                >>
ENTRY: FAILCD(1);
CAPACITY: 997;
<<                                                                >>
<<                 DATE FILE                                      >>
<<                                                                >>
NAME:  DATEF::43,A;        <<DATE FILE ALLOWS ACCESS BY DAY       >>
ENTRY: DATE(2);
           ^
UNDEFINED ITEM REFERENCED.
CAPACITY: 367;
         ^
MASTER MUST HAVE A PATH.
<<                                                                >>
<<                 REFERENCE DESIGNATOR FILE                      >>
<<                                                                >>
NAME:  REF::43,A;          <<REFERENCE DESIGNATOR FILE ALLOWS     >>
                           <<ACCESS BY DESIGNATOR                 >>
ENTRY: REFDES(1);
             ^
UNDEFINED ITEM REFERENCED.
CAPACITY: 997;
         ^
MASTER MUST HAVE A PATH.
<<                                                                >>
<<                 EMPLOYEE FILE                                  >>
<<                                                                >>
NAME:  EMPL::43,A;         <<EMPLOYEE FILE ALLOWS ACCESS BY EMP#>>
                           <<TO FAILURE FILE AND TIME VOUCHER     >>
                           <<FILE.                                >>
ENTRY: EMP#(2);
CAPACITY: 263;
<<                                                                >>
<<                 PROJECT NUMBER FILE                            >>
<<                                                                >>
NAME:  PROJ::43,M;         <<PROJECT NUMBER FILE HAS ASSOCIATED   >>
                           <<DEPARTMENT NUMBER AND OPENING AND    >>
                           <<CLOSING DATES OF NUMBER.             >>
ENTRY: PROJ#(1),
       DEPT#,
       OPDAT,
       CLDAT;
CAPACITY: 101;
<<                                                                >>
<<                 FAILURE FILE                                 >>
<<                                                              >>
NAME:  FAIL::43,D;         <<FAILURE FILE HAS A TRANSACTION     >>
                           <<ENTERED WHENEVER THE TEST TECH     >>
                           <<DETECTS A FAILURE                  >>
ENTRY: ASSY#(ASMBY),
       DATE(DATEF),
           ^

UNDEFINED ITEM REFERENCED.
       PART#(PART),
       FAILCD(CODE),
       REFDES(REF),
             ^

UNDEFINED ITEM REFERENCED.
       PNTS,
       EMP#(EMPL),
       SUP#(EMPL);
CAPACITY: 1009;
<<                                                              >>
<<                 TIME VOUCHER FILE                            >>
<<                                                              >>
NAME:  TIME::43,D;         <<TIME VOUCHER FILE CONTAINS EMPLOYEE>>
                           <<NUMBER, DEPARTMENT NUMBER, DATE AND>>
                           <<HOURS.                             >>
ENTRY: EMP#(EMPL),
       PROJ#(PROJ(DATE)),
       DATE(DATEF),
           ^

UNDEFINED ITEM REFERENCED.
       HOURS;
CAPACITY: 367;
END.


NUMBER OF ERROR MESSAGES: 0008
NUMBER OF ITEMS: 011
NUMBER OF SETS: 05
ROOT FILE:  00269 WORDS,  00005 BLOCKS

CARTRIDGE NUMBER                        NUMBER BLOCKS REQUIRED
        00043                                0000000686

ROOT FILE NOT CREATED - SCHEMA ERRORS.
END DATABASE DEFINITION
```

Figure 2-2. DBDS Run With Errors

DBDS General Messages
~~~~~~~~~~~~~~~~~~~~~

After DBDS processes a schema, it prints a group of general messages informing the user about the size of the root file, number of errors encountered, etc. These messages can be seen in the example above. The Root File mentioned in the summary is the file created by DBDS to contain structural information. For more information about the contents of the Root File, see Chapter 7.

Sample Data Schema

Figure 2-3 shows the complete schema for the QA Database.
~~~~~~~~~~~~~~~~~~~~~

```
$CONTROL:;
BEGIN DATABASE: QA:100:43;
LEVELS:
       1    OPER;
       5    ACCT;
      10    ADMIN;
      15    SECUR;
ITEMS:
       ASSY#,   X10(1,10); <<ASSEMBLY NUMBER FOR ASSEMBLY MASTER>>
                           <<AND FAILURE FILE.                   >>
       FAIL#,   I1(1,10);  <<NUMBER OF FAILURES - ASSEMBLY       >>
       PART#,   X8(1,10);  <<PART NUMBER FOR PART NUMBER MASTER  >>
                           <<AND FAILURE FILE                    >>
       FAILCD,  X4(1,10);  <<FAIL CODE                           >>
       REFDES,  X4(1,10);  <<REFERENCE DESIGNATOR                >>
       DATE,    X6(1,1);   <<DATE FOR FAILURE FILE, DATE MASTER  >>
                           <<AND PRODUCT TESTED FILE             >>
       EMP#,    X6(1,10);  <<EMPLOYEE NUMBER FOR EMPLOYEE MASTER>>
                           <<FAILURE FILE AND TIME VOUCHER FILE  >>
       SUP#,    X6(1,10);  <<SUPERVISOR NUMBER FOR FAILURE FILE  >>
       PROJ#,   X6(1,5);   <<PROJECT NUMBER FOR PROJECT NUMBER   >>
                           <<MASTER AND TIME VOUCHER FILE        >>
       DEPT#,   X4(1,5);   <<DEPARTMENT NUMBER                   >>
       OPDAT,   X6(1,5);   <<OPEN DATE FOR PROJECT NUMBER        >>
       CLDAT,   X6(1,5);   <<CLOSE DATE FOR PROJECT NUMBER       >>
       PNTS,8   X1(1,1);   <<FAIL/NO FAIL ARRAY FOR EACH PIN     >>
       HOURS,   R2(1,1);   <<HOURS PER PROJECT NUMBER            >>
SETS:
<<                                                               >>
<<                 ASSEMBLY FILE                                 >>
<<                                                               >>
NAME:  ASMBY::43,M;        <<ASSEMBLY FILE CONTAINS THE ASSEMBLY>>
                           <<NUMBER AND TOTAL NUMBER OF FAILURES>>
                           <<FOR THAT ASSEMBLY                   >>
ENTRY: ASSY#(1),
       FAIL#;
CAPACITY:409;
<<                                                               >>
<<                 PART NUMBER FILE                              >>
<<                                                               >>
NAME:  PART::43,M;         <<PART FILE CONTAINS ONLY PART NUMBER>>
                           <<NOTE THAT ALTHOUGH IT CONTAINS ONLY>>
                           <<ONE ITEM, IT IS A MANUAL MASTER SO >>
                           <<ENTRIES CAN BE CONTROLLED.          >>
ENTRY: PART#(1);
CAPACITY: 4001;
<<                                                               >>
<<                 FAIL CODE FILE                                >>
<<                                                               >>
NAME:  CODE::43,A;         <<FAIL CODE FILE CONTAINS ALL FAILURE>>
                           <<CODES                               >>
```

```
ENTRY: FAILCD(1);
CAPACITY: 997;
<<                                                         >>
<<                DATE FILE                                >>
<<                                                         >>
NAME:  DATEF::43,A;       <<DATE FILE ALLOWS ACCESS BY DAY     >>
ENTRY: DATE(2);
CAPACITY: 367;
<<                                                         >>
<<                REFERENCE DESIGNATOR FILE                >>
<<                                                         >>
NAME:  REF::43,A;         <<REFERENCE DESIGNATOR FILE ALLOWS   >>
                          <<ACCESS BY DESIGNATOR               >>
ENTRY: REFDES(1);
CAPACITY: 997;
<<                                                         >>
<<                EMPLOYEE FILE                            >>
<<                                                         >>
NAME:  EMPL::43,A;        <<EMPLOYEE FILE ALLOWS ACCESS BY EMP#>>
                          <<TO FAILURE FILE AND TIME VOUCHER   >>
                          <<FILE.                              >>
ENTRY: EMP#(3);
CAPACITY: 263;
<<                                                         >>
<<                PROJECT NUMBER FILE                      >>
<<                                                         >>
NAME:  PROJ::43,M;        <<PROJECT NUMBER FILE HAS ASSOCIATED >>
                          <<DEPARTMENT NUMBER AND OPENING AND  >>
                          <<CLOSING DATES OF NUMBER.           >>
ENTRY: PROJ#(1),
       DEPT#,
       OPDAT,
       CLDAT;
CAPACITY: 101;
<<                                                         >>
<<                FAILURE FILE                             >>
<<                                                         >>
NAME:  FAIL::43,D;        <<FAILURE FILE HAS A TRANSACTION     >>
                          <<ENTERED WHENEVER THE TEST TECH     >>
                          <<DETECTS A FAILURE                  >>
ENTRY: ASSY#(ASMBY),
       DATE(DATEF),
       PART#(PART),
       FAILCD(CODE),
       REFDES(REF),
       PNTS,
       EMP#(EMPL),
       SUP#(EMPL);
CAPACITY: 1009;
<<                                                         >>
<<                TIME VOUCHER FILE                          >>
<<                                                           >>
NAME:  TIME::43,D;          <<TIME VOUCHER FILE CONTAINS EMPLOYEE>>
                            <<NUMBER, DEPARTMENT NUMBER, DATE AND>>
                            <<HOURS.                             >>
ENTRY: EMP#(EMPL),
       PROJ#(PROJ(DATE)),
       DATE(DATEF),
       HOURS;
CAPACITY: 367;
END.
```

Figure 2-3. Sample Schema

Loading The Database

Once the database (root file and data sets) is created, it is ready to be loaded with data. This can be done in one of two ways.

- Running a utility program DBBLD, with bulk input
- Writing a FORTRAN, or Assembly Language program to add data on-line and/or from batch input.

The operation and file format for DBBLD is described below. A discussion of database access through program calls will be found in Chapters 4 and 5.

DBBLD

DBBLD loads actual data item values into a database structure from tape or disc file. Besides storing data, DBBLD confirms that the data is presented in the proper format. DBBLD is useful for initially storing large amounts of data into a newly-created database or adding data entries to data already stored in a database. The data can be prepared in a disc file, checked and edited before being conveniently loaded by DBBLD.

Preparing Data For DBBLD

DBBLD reads a source file consisting of physical records. When running DBBLD, the user can specify an integer n, which will be the number of columns in each source file record containing data. The remainder of the record is ignored by DBBLD and can be used for sequencing information.

The entire database file is identified by an initial record signifying the name of the database, its cartridge number and security code, and if any levels are defined in the level part of the database schema, some level code word. This level code word must be high enough to allow write access to any item for which the user wishes to add a value. The record starts in column 1 and assumes the following format:

```
database name:security code[:cartridge number][,level word];
```

For example, the first record might be

```
     QA:100:43,SECUR;
```

Individual data entries must be grouped according to the data set to which they belong. Data entries for a specific data set are identified by a set record. This record takes the format

```
     $SET:setname
```

where

setname   is the name of the data set into which the data is to be inserted.

The data entries do not have to be in any particular order. However, data entries for a manual master data set must appear before the data entries for any detail data sets linked to it.

Columns 1 through n of each data record are divided into fields. Each one of these fields contains a data item value corresponding to a data item of the data set specified in the set record. Data item values must appear in the data entry in the exact order that they are listed in the ENTRY: statement of the database schema set part for that data set. For example,

```
     NAME:  PROJ::43,M;
     ENTRY: PROJ#(1),
            DEPT#,
            OPDAT,
            CLDAT;
     CAPACITY: 101;
```

is the set part for the data set PROJ. A data entry for this data set must first contain a value for PROJ#, then DEPT#, OPDAT and CLDAT. All the fields pertaining to a data entry must be joined together without intervening spaces in the data file records. The field for the first data item in each data entry must start in column one of a data record. The fields continue through column n of the record unless a field extends beyond column n. In that case, the field must start in column one of the next record. If a field's length is greater than the record length, the field must start in column one of a record, extend through column n and continue with column one of the next record. An example of a data entry extending over records is shown below.

Assume a record length of 60 characters. An entry is defined as follows:

ITEM1	X10
ITEM2	X20
ITEM3	X10
ITEM4	X50
ITEM5	X80

For each entry to be entered with DBBLD, the following series of records must be presented.

```
+------------------------------------------------------------+
|        |                   |          |                    |
| ITEM1  |       ITEM2       | ITEM3    |      no data       |
|        |                   |          |                    |
+------------------------------------------------------------+
1        10                  30         40                   60

+------------------------------------------------------------+
|                                                   |        |
|               ITEM 4                              | no data|
|                                                   |        |
+------------------------------------------------------------+
1                                                   50       60

+------------------------------------------------------------+
|                                                            |
|               ITEM5 (first sixty characters)               |
|                                                            |
+------------------------------------------------------------+
1                                                            60

+------------------------------------------------------------+
|                  |                                         |
|       ITEM5      |                 no data                 |
|      (con't)     |                                         |
+------------------------------------------------------------+
1                  20                                        60
```

Not every data item need receive a value when entering data into the data base. In this case, the field can be left blank.

The field size is determined by the data type of the data item. I1 (integer) type data item values must appear in a six column field, while R2 (real) type data item values appear in a thirteen column field. X (character) type data item values appear in a field exactly as long as the defined length.

Compound item values must appear in order and there must be a field for each element in the item.

Type I1 data item values must be in the range -32768 to +32767. When a number is negative, the user must insert a minus sign to the left of the number. A plus sign or a blank indicates a positive number. I1 data items may appear anywhere within the six column field. To omit entering a value, simply leave the field blank and DBBLD will enter the value 0 for that item.

R2 type data item values can be entered either as fixed or floating point numbers in the range of ±1.70 * 10**38 to ±1.47 * 10**-39. Floating point numbers take on the format

    nE+m   or   nE-m

where:

n is a fixed point number with eight digit accuracy.
m is the power of 10 n is to be multiplied by.

Negative real numbers are denoted by a minus sign (-) preceding the number. A real number may appear anywhere within a thirteen column field. Table 3-1 shows some examples of real number entries.

Table 2-1. Real Number Entries

Real numbers to be entered	DBBLD field contents
3.1416	3.1416
524.67	5.2467E+2
-77.985	-77.985
.00352	3.52E-3
.12364	.12364
-.0075	-7.5E-3

To omit entering a value for a real number, leave the field blank and DBBLD will enter the value zero for the item.

X type data item values can contain any ASCII characters. Leading and trailing blanks in a character-valued field are significant and count as part of the data item value.

For example, ASSY# is a data item defined as X10. The value

```
     92060111^^
```

is not the same as

```
     ^^92060111 (^=ASCII BLANK)
```

Character value items can be used to hold numeric values and arithmetic operations can be carried out on these items after they have been entered into the database.

The last record of the DBBLD input file must be

```
     $END
```

starting in column one.

Running DBBLD

To execute DBBLD enter the following command:

```
     :RU,DBBLD,input,list,options
```

where:

input	is the FMP file namr which contains the data to be placed in the database. The default is the scheduling lu.
list	is the FMP namr to which the list is to be written. All output and errors will be listed to the list file or device. The default is LU 6.
options	is a list of the following options entered in any order separated by commas: ADD,ERRHLT,NOLIST,n

where:

ADD	indicates that entries in the input file are to be added to already existing entries in the database. Although all entries are processed only those error free items are added to the database. When ADD is not specified all the entries are processed, checking for errors, but none are added to the database.
ERRHLT	indicates that all processing is to halt when an error is detected. If ERRHLT is not specified, the entire input file is processed regardless of any errors encountered.

NOLIST	indicates that data from the input file is not to be listed to the list device. When NOLIST is omitted, each input record is listed.
n	is an integer from 1 to 512, inclusive, which specifies the input line length. Default is 72.

DBBLD then proceeds to read the data file. If the user requested a listing, the data is listed as it appears in the file. If an error occurs, a message is printed under the offending record in the listing. If no listing was requested and an error occurs, DBBLD prints the the offending record followed by the error message. See Appendix B for help in interpreting the error messages.

When entering large amounts of data into a database, it is strongly suggested that you run a test containing values for only one entry for each data set in the database. By not including the ADD parameter in the directive RU,DBBLD, you can confirm that the data fields have been correctly entered in the DBBLD input file prior to loading data into the database.

Upon completion, DBBLD prints two messages upon the list device. The first message

```
NUMBER OF ERRORS:0000
```

informs the user of the errors encountered during the processing of the data input file. The second message

```
DATABASE SUCCESSFULLY BUILT OR UPDATED
```

informs the user that the creation or update procedure was successfully completed. This occurs only if there were no errors. The message

```
DBBLD STOP: 0000
```

appears on LU 1 upon completion of DBBLD.

Sample DBBLD Run

Figure 2-4 shows the data input file for the database schema previously shown in Figure 2-3 and Figure 2-5 shows the output listing created by DBBLD (the default options were used).

```
QA:100:43,SECUR;
$SET:ASMBY
92060160010
92060180010
92060600010
92060800010
92087160010
92087180010
92087600010
92087800010
92088160010
92088180010
92088600010
92088800010
$SET:PART
59501111
59501134
59501460
59501489
59501550
$SET:PROJ
9400715210781121
9400725210781121
9400735210781121
9400745210781121
9400815230781121
9400825230781121
9400835230781121
9400845230781121
9400855230781121
9400915260781121
9400925260781121
9400945260781121
9401025290781121
9401045290781121
9401055290781121
```

```
:SET:FAIL
:20601800178122859501111103 K23
920601800178120159501111202 L10
92060180017812015950111l
920601800178120159501111
920601800178120159501111
920601800178120159501111
920601600178120759501550102 K12
920601600178120759501550103 K12
920601600178120759501111102 K12
920601800178120759501134109 K41
920601880178120759501134203 J42
920606000178120759501489203 K12
920608000178120759501489105 J14        45003
920601600178122859501489103 K9
920608000178122859501134201 L17        45003
920608000178122859501550307 H12        45003
920606000178120159500001202 L10
920601600178120259501489305 J13 000011114322l
$END
```

Figure 2-4. Data Input for DBBLD

```
IMAGE/1000 DATABASE BUILD UTILITY

QA:100:43,SECUR;
$SET:ASMBY

  ASSY#  IN COLUMNS 0001 THROUGH  0010 IS TYPE X
  FAIL#  IN COLUMNS 0011 THROUGH  0016 IS TYPE I

12345678901234567890123456789012345678901234567890123456789012345678901234567890
92060160010
92060180010
92060600010
92060800010
92087160010
92087180010
92087600010
92087800010
92088160010
92088180010
92088600010
92088800010
$SET:PART

  PART#  IN COLUMNS 0001 THROUGH  0008 IS TYPE X

12345678901234567890123456789012345678901234567890123456789012345678901234567890
59501111
59501134
59501460
59501489
59501550


$SET:PROJ

  PROJ#  IN COLUMNS 0001 THROUGH  0006 IS TYPE X
  DEPT#  IN COLUMNS 0007 THROUGH  0010 IS TYPE X
  OPDAT  IN COLUMNS 0011 THROUGH  0016 IS TYPE X
  CLDAT  IN COLUMNS 0017 THROUGH  0022 IS TYPE X


12345678901234567890123456789012345678901234567890123456789012345678901234567890
9400715210781121
9400725210781121
9400735210781121
9400745210781121
9400815230781121
9400825230781121
9400835230781121
9400845230781121
9400855230781121
9400915260781121
9400925260781121
9400945260781121
9401025290781121
9401045290781121
9401055290781121
$SET:FAIL
9206018001781228595O1111
920601800178120159501111
920601800178120159501111
920601800178120159501111
920601800178120159501111
920601800178120159501111
9206016001781207595O1550102 K12
920601600178120759501550103 K12
920601600178120759501111102 K12
920601800178120759501134109 K41
920601800178120759501134203 J42
920606000178120759501489203 K12
920608000178120759501489105 J14         45003
920601600178122859501489103 K9
920608000178122859501134201 L17         45003
920608000178122859501550307 H12         45003
920606000178120159501111202 L10
920601600178120259501489305 J13 0000111143221
$END
 NUMBER OF ERRORS:0000
```

Figure 2-5. Sample DBBLD Run.

Chapter 3
Querying The Database

```
+-----------------------------------------+
|                  NOTE                   |
|                                         |
|  QUERY is only available to 92073A users |
|  by running remotely  from a database on |
|  an RTE-IVB node.  Refer to the DS/1000  |
|  Network  Manager's  Manual  for  remote |
|  access information.                     |
+-----------------------------------------+
```

QUERY provides a simple method of accessing an IMAGE database without programming effort. QUERY may be used to do the following:

- store data
- modify or delete data values on-line
- retrieve data which meets selection criteria
- report on the data retrieved

These operations may be performed by entering simple commands consisting of English-language words such as FIND and REPORT.

The structure of the disc files need not be known, only the relationships of the database elements. QUERY finds the data and performs the operations in response to commands using the data set and data item names specified.

Commands can be submitted to QUERY either interactively or from a batch file. If the input to QUERY is a batch file, each record in the file should be that information expected after an interactive prompt. QUERY may also be scheduled to run at a remote system if a Distributed Systems (DS/1000) link is available. The remote site would be the location of the database to be accessed.

QUERY's report formatting capability allows the building of reports with header and column labels, page numbers, and group labels. In addition, entries can be sorted on multiple fields, and QUERY can report total and average values or count occurrences of data values.

A frequently used or complex command can be stored as an individual procedure in an command file (known as a procedure file). The procedure name can then be used in place of the command parameters. Often repeated sequences of commands can be stored in a batch file and executed while running QUERY interactively.

Information about the database structure is available with the FORM command and information about QUERY commands, their function, format, and parameters is available with the HELP command.

All of the tasks described above can be accomplished without programming. QUERY makes an excellent debugging aid in developing programs which access IMAGE databases through the IMAGE library subroutines. Data base contents can be altered using a program and then QUERY can examine the data to determine the results of the programmed changes.

Commands

QUERY commands are described in detail in the following Chapters. However, you should understand these characteristics which apply to all commands:

- Command names must be entered as specified in the command descriptions.
- Commands consist of English words and parameters (both required and optional) separated by commas or semi-colons, depending on the command syntax.
- Each command input must end with a semi-colon or a zero-length record.

Data Item Elements

In the definition of the database, IMAGE allows the user to specify compound items which can contain more than one element. Each element in the item may have a value. QUERY locates and processes only the first element in a compound item with the FIND and REPORT commands. REPORT ALL and UPDATE will process all elements in a compound item

Qualified Data Item Names

IMAGE allows the database designer to use the same name for two or more items provided the items are not part of the same data set. If you are referring to such an item, you must specify which data set to access. This is done by qualifying the item name with the data set name. A qualified data item name is a data set name followed by a period, followed by a data item name. For example,

```
     LABOR.BADGE
```

BADGE is the name of a data item in the data set named LABOR. If BADGE is also the name of a data item in a data set named EMPLOYEE, its qualified name for that data set will be EMPLOYEE.BADGE.

Using Query

To request QUERY from a terminal, the user types the directive

```
     :RU,QUERY,input,list,log,ECHO,node
```

where:

input — is the FMP namr from which the input will come. The default is the user's terminal in an MTM environment or LU 1.

list — is the FMP namr where the list will be written. The default list is the interactive input device or LU 6 if the input device is a file or non-interactive. When list is a file which does not already exist, it will be created.

log — specifies the logging device. Errors are always logged to the log device. When specified by the ECHO option, each command will be listed to the log device before it is executed. If ECHO is not specified, the commands are not listed to the log device. The default log unit is the interactive input device or LU 1 when the input unit is a file or non-interactive.

node — allows the user to run QUERY remotely using the DS REMAT program. Node is the user's node number. When a node number is specified, the input device must be either interactive or an FMP file namr. All lus specified must be system, rather than session lus. All files and databases referenced by QUERY running remotely must exist on the node local to the QUERY program. Before requesting a remote copy of QUERY, the REMAT switch (SW) command must be executed with the source node as the node at which QUERY is to be run. The remote copy of QUERY can be invoked using the RW REMAT command. For information on using REMAT, see the DS Programmers Reference Manual.

When the input is from an interactive device, QUERY will print its heading and prompt for user response. For example:

```
QUERY READY
NEXT?
```

When the input is from a batch file, the prompting is suppressed and input is expected in the order that it would be entered interactively. Each batch record must contain only the information that would be entered after an interactive prompt. An error in the command stream will cause QUERY to terminate.

Any input to QUERY can be typed on any number of lines. The maximum length of a line is 72 characters. QUERY will not expect another command until a semi-colon followed by a carriage return is encountered or until a zero length record is entered. This means when an interactive user enters only a carriage return, QUERY will assume the input record is complete. Only 1786 characters may be entered in one command.

QUERY will respond to the RTE BR[EAK] command during a REPORT FIND or UPDATE command, unless it is running remotely.

Opening A Database For Use

Immediately after requesting QUERY with a directive, the user must define the environment to QUERY; which database is to be accessed, and which files are to be used for retrieving data. This information is given to QUERY through the DATA-BASE= and SELECT-FILE= commands.

Once the user enters one of these two commands, the file is opened to the user until another DATA-BASE= or SELECT-FILE= command is entered or until execution of QUERY is terminated. When the user invokes QUERY again, new commands to inform QUERY of the data-base and select-file must be entered.

Defining A Data-Base

The DATA-BASE= command informs QUERY which database QUERY is to access. The format of the command is

```
     DATA-BASE=database name;
```

where:

database name	is the FMP file namr of an IMAGE database root file created by DBDS. The FMP file namr must contain the file name and the security code. The cartridge number is optional.

When the input is interactive, QUERY will then prompt the user for the level code word and the open mode. In batch mode, QUERY will expect the code word followed by a semi-colon and the open mode followed by a semi-colon, as the next two input records.

QUERY's prompt for the level code word is as follows:

```
     LEVEL=?
```

The user responds with the level word appropriate for the level of access needed. The level words are defined in the schema definition when creating the database. The level word will determine which items are accessible by the user.

Finally, QUERY asks for the mode of access to the database by typing

```
     OPEN MODE = ?
```

The user's response depends upon the intended use of QUERY, as shown below.

MODE	ACCESS
1	Read and write, shared access
3	Read and write, exclusive access
8	Read only, shared access

Defining A Select-File

During FIND, UPDATE and REPORT operations, QUERY uses a scratch file called a select file to keep relative record numbers. This file must be defined before any FIND, UPDATE REPLACE, UPDATE DELETE or REPORT operations can be performed on the database. The FIND command will cause numbers to be placed in this file. UPDATE and REPORT commands will use the current contents of the select file to determine what records to use in their operations. The select file may have been created before entering QUERY. If not previously created, QUERY will create it.

User programs may also use the select file to access records which had previously been selected by a QUERY FIND. For more information on the format of the select file, see Chapter 7.

The format of the command is:

```
     SELECT-FILE=file name;
```

where:

file name is the FMP file namr of a disc file.

If the file named does not exist, it will be created. If it already exists, it must be a type 1 file at least three blocks long. Each time a FIND command executes, the SELECT-FILE is re-used, allowing the same file to be used repeatedly.

The user can change the select file any time QUERY prompts for a command. The SELECT-FILE= command need only be given once in a QUERY session, unless the user desires to change files in order to save the contents of the current select file. In the example shown below both SEL and SELFIL contain record addresses of data entries retrieved by a QUERY FIND command.

```
     NEXT?DATA-BASE=QA:100:43;
     LEVEL = ?ADMIN;
     OPEN MODE = ?1;
     NEXT?SELECT-FILE=SEL;
     NEXT?FIND FAIL.ASSY# IS "9206018001" END;
        0000000008 ENTRIES QUALIFIED
     NEXT?SELECT-FILE=SELFIL;
     NEXT?FIND ASMBY.ASSY# IS "9206080001" END;
        0000000001 ENTRIES QUALIFIED
```

Using Procedure And Command Files

FIND, UPDATE and REPORT commands allow the use of procedure files which allow the user to store in a disc file procedures for finding, updating or reporting. There is only one command per procedure file. When using procedure files to present command information, the syntax is:

```
     command NAME=procedure name
```

where:

command	is either REPORT, UPDATE or FIND
procedure name	is the FMP file namr of the file containing the procedure

Procedure files are distinguished from command files, which are used in conjunction with the XEQ command. Command files can contain many commands plus the parameters required by those commands. Procedure files are limited to one command per file.

One of the most important uses of procedure files is with the REPORT command, since a REPORT command consists of many statements. Rather than entering each statement every time a particular report is desired, a report procedure file can be created that can be used whenever the report is needed.

When QUERY is being executed remotely, all procedure and command files must be local to the node at which QUERY is executing.

Retrieving Database Entries

A main feature of QUERY is the capability to locate data entries in the database according to the values of data items in the entry. The user accomplishes this using the FIND command. The FIND command includes the following features:

- The FIND command can search up to fifty different data items in a single data set.
- FIND commands can be stored as procedures on the disc (using the CREATE command) for repeated use without retyping.

- FIND procedures can be created which prompt the user for the desired search values at execution time, allowing the user to search the same data items for different values each time the procedure is run.
- The FIND command can be used to search for multiple values of the same data item.
- The FIND command reports how many data entries were discovered which fulfill the criteria of the FIND command.

Find Command

The FIND command retrieves data entries from the database by storing the record addresses of the selected entries in the current select file.

The format of the FIND command is:

```
     FIND retrieve procedure END;
```

or

```
     FIND NAME=procedure name;
```

where:

retrieve procedure is a group of data item names, data item values, and relational operators joined together by logical connectors.

procedure name is the FMP file namr of a file stored on disc containing a FIND command using a retrieve procedure.

The FIND command is used in conjunction with the REPORT and UPDATE commands. The user locates data entries in the database through the criteria specified in the FIND command, and modifies or reports the entries through the UPDATE or REPORT commands.

Retrieve Procedures

The retrieve procedure in a FIND command specifies a relation or a series of relations between a data item and a data item value. QUERY compares the relation defined in the retrieve procedure with the values of the data items in the data entries and stores the record addresses of those data entries which satisfy the relationship in the current select file. The form of the relation is:

```
[set name.] item name operator "value"
```

where:

set name	is the name of a data set which contains the item. The set name is optional when the item is defined in only one set, or when the data set is specified earlier in the FIND command. Only one set is searched at a time and the set to be searched is determined, explicitly or implicitly, by the membership of the first item named.
item name	is the name of an item. When the item name specifies a compound item, only the first element is used.
operator	is the relational operator and indicates the type of comparison to be made. The relational operators are shown in Table 3-1.
value	is enclosed in quote marks and consists of a value to be compared with each of the values of the data items named in the relationship. The value should be appropriate to the data item type.

Table 3-1. Retrieve Procedure Relational Operators

RELATIONAL OPERATOR	MEANING
IS IE	equals
ISNOT INE	is not equal to
ILT	is less than
INLT	is not less than
IGT	is greater than
INGT	is not greater than

For example, if the user wishes to find data entries with a data item called REFDES whose value is equal to "J14", he enters the following FIND command:

```
FIND FAIL.REFDES IS "J14" END;
```

If the user wishes to find data entries with a data item called DATE whose value is less than 781201, he enters the following FIND command:

```
FIND FAIL.DATE ILT "781201" END;
```

To make more than one comparison for each data entry selected, the user connects two relationships with the logical connector AND or OR. When an AND connector is used, QUERY selects only those data entries whose data item values satisfy both relationships on either side of AND. When an OR connector is used, QUERY selects those data entries that satisfy the conditions on either or both sides of the OR.

For example, the FIND command

```
FIND ASSY# IS "9206060001" AND DATE IS "781201" END;
```

instructs QUERY to retrieve only those data entries with the data item ASSY# equal to "9206060001" and the data item DATE equal to "781201" at the same time. QUERY does not select a data entry just because the value of ASSY# is "9206060001"; the value of DATE for that entry must be "781201" as well.

The command

```
FIND FAIL.FAILCD IS "202" OR DATE IGT "781031" END;
```

instructs QUERY to retrieve data entries with data item FAILCD equal to "202" and to retrieve data entries with data item DATE greater than "781031". Only one of the relationships need be true for QUERY to select the data entry, although both can be true.

Retrieve procedures can contain both AND and OR connectors. In this case, QUERY satisfies the AND connector first. For example, the FIND command

```
FIND FAIL.FAILCD ILT "100" OR FAILCD IGT "500" AND
REFDES IS "L10" END;
```

retrieves all data entries that simultaneously contain the value of FAILCD greater than 500 and the value of REFDES equal to L10. QUERY also retrieves all data entries with the value of FAILCD less than 100, regardless of the value of REFDES.

Parentheses cannot appear in a retrieve procedure, but it is possible to construct forms which act as parentheses. For example, if Cn stands for a relationship such as REFDES IS "K19" then

```
     (C1 OR C2) AND C3
```

is represented as

```
     C1 AND C3 OR C2 AND C3
```

If a relationship defined in a retrieve procedure uses the relational operators IS, IE, ISNOT or INE, the user can request that QUERY search for more that one value for a data item. To do so, the user specifies the values one after the other separated by commas.

For example, the following FIND command requests that QUERY retrieve all data entries with the value of data item DATE equal to "781201" or equal to "790101":

```
     FIND FAIL.DATE IS "781201", "790101" END;
```

The FIND command above is also equivalent to the FIND command

```
     FIND FAIL.DATE IS "781201" OR DATE IS "790101" END;
```

Preceding the data item name by a data set name, separated by a period, qualifies a data item whose name appears in more than one data set of the database. Qualifying the data item name in this way tells QUERY which of the data sets is to be used in searching for entries which satisfy the remainder of the relation.

For example, assume "DATE" to be an item in both of the two sets "FAIL" and "TIME", then

```
     FIND FAIL.DATE IS "781201" END;
```

instructs QUERY to retrieve all entries in the data set "FAIL" with the data item "DATE" equal to "781201". The command

```
     FIND TIME.DATE IS "781201" END;
```

instructs QUERY to retrieve all entries in the data set "TIME" which satisfy the same requirements. It is not possible to search both FAIL

After the user types a FIND command, QUERY may type the following message:

```
     SERIAL READ BEING PERFORMED
```

QUERY types this message whenever a serial search must be performed of a data set to retrieve the qualifying data entries. A serial search must be performed whenever a non-key item is being searched for, or whenever a relational operator other than IS or IE is used. This means that QUERY must search each entry in the data set without the benefit of a chained or hashed read.

After a FIND command has been successfully executed, QUERY reports the number of data entry addresses that are now stored in the select file by typing the following message.

```
     0000000000 ENTRIES QUALIFIED
```

The number indicates the number of data entries found which satisfy the conditions set forth in the FIND command. If no entries qualify, then the number is equal to zero.

Find Procedures

A FIND procedure stored on the disc can be created so that QUERY prompts the user for the values to be compared with data entries in the data set. To do this, the user types null values (quote marks with no intervening characters) in the place of the data value. When QUERY executes the FIND procedure (in response to a FIND NAME=procedure name command) QUERY prompts the user to type in a value for each set of null values in the FIND procedure. For example, the FIND procedure:

```
     FIND FAIL.REFDES IS "" END;
```

causes QUERY to type the following message when the procedure is executed:

```
     WHAT IS THE VALUE OF REFDES?
```

The user types the desired value of REFDES enclosed in quotation marks following the question mark. QUERY then searches the database and retrieves any data entries which satisfy the relationship in the retrieve procedure.

If the user wishes to compare multiple values, he writes the retrieve procedure with multiple null values. QUERY prompts for each set of quote marks in the retrieve procedure. For example, the FIND procedure:

```
FIND FAIL.REFDES IS "","" AND FAILCD IE "","" END;
```

causes QUERY to print the following messages when the FIND procedure is executed.

```
WHAT IS THE VALUE OF REFDES?"L10";
WHAT IS THE VALUE OF REFDES?"K11";
WHAT IS THE VALUE OF FAILCD?"101";
WHAT IS THE VALUE OF FAILCD?"202";
   0000000002 ENTRIES QUALIFIED
NEXT?
```

The user types in the desired values to be used in retrieving data entries following the question mark. Values typed must not exceed 255 characters and should be appropriate to the data item type. When input is from a batch file, QUERY will not print the prompt message, but will expect the item values in consecutive records in the input file.

Modifying The Database

The UPDATE command is used to modify the database. The QUERY user can modify the database in one of three ways.

- The UPDATE ADD or UPDATE A command adds a data entry to a detail or manual master data set.
- The UPDATE DELETE or UPDATE D command deletes one or more entries from a manual master or detail data set.
- The UPDATE REPLACE or UPDATE R command replaces values for a specific data item in one or more data entries with a new value specified by the user, allowing the user to update data in the database.

When the user specifies an UPDATE A command, QUERY prompts the user by printing the names of the data items in the data entry. The user types in the data item value following the data item name.

To use the UPDATE D or UPDATE R command the user must first use the FIND command. The FIND command stores the record addresses of the data entries the user desires to delete or modify in the select file. When UPDATE D or UPDATE R is typed, QUERY operates on all the data entries specified in the select file.

The UPDATE command can be specified to QUERY in one of two forms: the user can either type the entire command on the keyboard device, or specify the name of an UPDATE procedure previously defined and stored on the disc.

To use the UPDATE command, the user must open the database in mode 1 (shared read/write access) or mode 3 (exclusive read/write access).

Adding Data

The UPDATE A command adds a data entry to a detail or manual master data set.

The format of the command is:

```
     UPDATE A[DD],data set name;
```

or

```
     UPDATE NAME=procedure name;
```

where:

data set name is the name of the data set to which the entry is to be added.

procedure name is the namr of the UPDATE A command stored as a procedure on the disc.

When the user types an UPDATE A,data set name; command, QUERY prompts the user for individual item values by printing the name of each item in the order the items are defined in the data schema. The user types the value for the data item following the data item name. The user must have an access level that allows write access to each item in the data set.

The value entered must be the same type as the type defined for the data item in the data schema. If a data item value is entered with invalid characters, QUERY types an error message and prompts the user again for that item. If the data item is defined as a key or sort item, the user must enter a value, otherwise QUERY reprompts for the key or sort item until a value is entered. A user can terminate the UPDATE ADD command by breaking QUERY with the BReak command.

The user must enter data item values enclosed in quotes (quotes on each side of the value) or as a null value (if the data item is not a key item).

Elements of a compound item are entered in a list separated by commas, and terminated by a semi-colon. When two commas are adjacent a null value is entered for that element in the item. Each item element must be enclosed in quotes.

The following example shows an UPDATE command for an item which is a compound item. The second element of the item is defaulted to zero.

```
     NEXT? UPDATE A,COTEST;
     TSTNO=? "3";
     CO2TST=? "20",,"60";
     ERROR=? "-1.02E-06";
     NEXT?
```

The user enters a null value for a data item by entering a semi-colon, or blanks followed by a semi-colon. QUERY enters the value zero for numeric items, and blanks for ASCII items. If a key item is given a null value, QUERY will continue to reprompt for the item value until it is entered, or until a BReak command is entered. (A user may get the attention of the operator interface by entering a carriage return and any spurious character in rapid succession.) When an illegal value is entered for a data item or element of a compound item, an error message is printed and the user is prompted for the value again. When the value is a list, the whole list must be reentered.

Integer values must range between -32768 and +32767. Real values must range between $\pm$1.70 * 10**38 and $\pm$1.47 * 10**-39. Real values may be entered in fixed or floating format. All numeric values may be preceded by a plus or minus sign. When neither is supplied a positive value is assumed. For ASCII values, leading or trailing blanks are considered part of the value. When the ASCII value is shorter than the size of the item, QUERY left justifies the value and blank fills to the right.

To use the UPDATE NAME=procedure name; command, the user must first store an UPDATE A command as a procedure on the disc by using the CREATE command or the RTE Editor. Once the procedure is stored on the disc, the user executes the procedure by typing UPDATE NAME=procedure name. QUERY searches for the procedure on the disc. If the procedure does not exist, QUERY informs the user and prompts for another command. If the procedure is found, QUERY proceeds exactly as if the command were entered at the keyboard. Note that the values which need to be entered are still prompted for at the user's terminal and thus, must be entered there.

Deleting Data

The UPDATE D command deletes one or more data entries from a detail or manual master data set.

The format of the command is:

```
     UPDATE D[ELETE];
```

or

```
     UPDATE NAME=procedure name;
```

where:

procedure name is the namr of the UPDATE D command stored as a procedure on the disc.

The UPDATE D command operates only on data entries whose record addresses are stored in the select file (through the FIND command). When the user types an UPDATE D command, all data entries whose record addresses are stored in the select file are deleted from the database. For this reason, it is recommended that the user do a REPORT ALL operation on the select file before doing an UPDATE D operation to be sure that the records to be deleted are those that the user intended.

To use the UPDATE NAME=procedure name command, the user must first store an UPDATE D command as a procedure on the disc using the CREATE command or the RTE Editor. Once the procedure is stored, the user executes the procedure by typing UPDATE NAME=procedure name. QUERY searches for the procedure and executes it. If the procedure named in the command does not exist, QUERY informs the user and prompts for another command.

The user must have access to all data items in the entry in order to delete a data entry. When attempting to delete a data entry from a manual master data set, the user must take care that there are no detail data entries still linked to the master data entry. If the master key item value being deleted still does exist in the detail data set, QUERY prints an error message to the user and prompts for another command.

For example, Figure 3-1 shows part of a database containing a master and detail data set linked together by key item. If a user enters the QUERY command

```
     FIND PART.PART# IS "59501111" END;
```

QUERY places the record address of the master data set entry containing the value "59501111" for the item name PART# in the select file.

If the user then types the command

```
     UPDATE D;
```

QUERY ignores the command and prints an error message to the

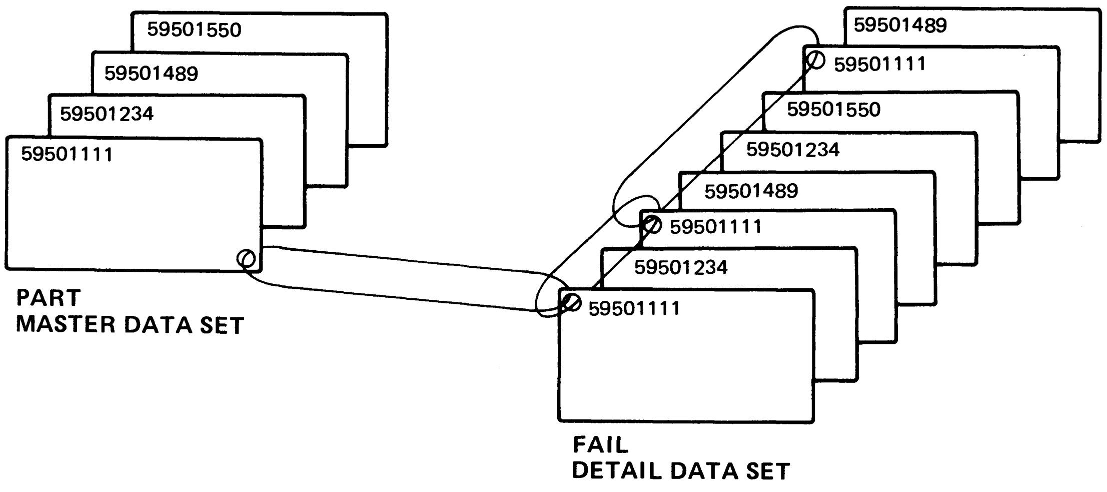


Figure 3-1. Deleting from a Master Data Set

user because the detail data set still contains data entries with the key item equal to "59501111". Automatic master data entries are automatically deleted when all of the detail data entries linked to it are deleted.

Care must be taken when attempting to delete more than one data entry from a master data set at one time. After QUERY deletes the first entry whose address is stored in the select file, under some circumstances, another data entry may be moved into the location just vacated by the deleted entry. If the previous address of the relocated data entry exists in the select file, QUERY finds that the location is now empty and returns an error message to the user.

To avoid this problem, the user should take one of the following courses of action: 1) delete only one master data entry at a time by storing only one entry in the select file (through the FIND command) 2) enter a new FIND command to locate the moved data entry after the error message is given or 3) retrieve the entries to be deleted by key item values. Below is an example of deletion of detail data entries from the data base using the UPDATE D command.

```
    NEXT?FIND FAIL.ASSY# IS "9206016001" AND FAILCD IS "305" END;
       0000000001 ENTRIES QUALIFIED
    NEXT?REPORT ALL;

    ASSY#  = 9206016001
    DATE   = 781202
    PART#  = 59501489
    FAILCD = 305
    REFDES = J13
    PNTS   = 0,0,0,0,1,1,1,1
    EMP#   = 43221

    NEXT?UPDATE D;
    NEXT?FIND FAIL.ASSY# IS "9206016001" AND FAILCD IS "305" END;
       0000000000 ENTRIES QUALIFIED
    NEXT?
```

Replacing Data

The UPDATE R command replaces the value of a data item with another value in all data entries in the current select file.

The format of the command is:

```
    UPDATE R[EPLACE];
```

or

```
    UPDATE NAME=procedure name;
```

QUERY then prompts the user for a data item to replace as follows.

```
    ITEM?
```

The user must enter the name of a data item and its value as follows.

```
    item name="value";
```

where

item name is the name of an item which is defined in the data set specified in the FIND command. It must not be a key or sort item. IMAGE does not allow key or sort items to be modified. The user must have write access to the item.

value is an appropriate value for the item's type. Compound items may be entered as a list of values, each value enclosed in quotes, separated by commas, terminated by a semi-colon.

When all items have been entered, the user enters a semi-colon to the REPLACE prompt, as follows.

```
     ITEM?;
     NEXT?
```

The UPDATE R command operates only on data entries whose record addresses are stored in the select file through the use of a FIND command. When a user types an UPDATE R command, the value of the data item(s) mentioned in the command is replaced with the new value in every data entry specified in the select file.

To use the UPDATE=procedure name command, the user must first store an UPDATE R command as a procedure on the disc by using a CREATE command or the RTE Editor. This file must contain a data item name and value for each item that is to be replaced. Once the procedure is stored, the user executes the procedure by typing UPDATE NAME=procedure name;. QUERY searches for and executes it. If the procedure named in the command does not exist, QUERY informs the user and prompts for another command by printing NEXT?. A procedure file must contain the item names and their values. Item name = value must follow the UPDATE R; command in successive records of the procedure file.

QUERY does not allow the user to update key item values. Any attempt to do so results in an error message.

The following is an example of modifying data item values in a data set using the UPDATE R command.

```
     NEXT?FIND ASMBY.ASSY# IS "9206016001" END;
        0000000001 ENTRIES QUALIFIED
     NEXT?REPORT ALL;
```

```
ASSY#   = 9206016001
FAIL#   = 3

NEXT?UPDATE R;
ITEM?FAIL#="4";
ITEM?;
NEXT?REPORT ALL;

ASSY#   = 9206016001
FAIL#   = 4

NEXT?
```

It is advisable to do a verification of the data entries in the select file before executing an UPDATE R command to be sure that only those entries to be changed have been selected. This is easily done, as shown in the above example, by using the REPORT ALL command.

Generating Reports

The REPORT command instructs QUERY to print information (on the list device or file) about data entries whose record addresses are stored in the current select file by a FIND command.

The REPORT command can be used on several levels. The user can either ask QUERY to print the name and value of each data item for all the data entries specified in the select file or request the data item values for all of the data entries without printing the data item names. Finally, the user can create output formats, complete with page headings, page numbers, subtotals and totals of data item values. The REPORT command can be stored as a procedure in a file through the use of the CREATE command or the RTE Editor and can then be used over again without retyping the report format.

Other features of REPORT include:

- Editing functions which allow the suppression of leading zeroes, insertion of dashes (-), dollar signs ($), commas (,), decimal points (.) and other characters.
- Counting, averaging and totaling functions which count the number of data entries reported, average the values of a data item in the report, or total the value of a data item.
- Spacing and page ejection functions are also controlled by the REPORT command.

Report Command

The REPORT command prints information about data entries specified in the select file by means of the FIND command.

The format of the command is:

```
     REPORT ALL [,character];
```

or

```
     REPORT NAME=procedure name;
```

or

```
     REPORT [,character];
     body
     END;
```

where

character	is any ASCII printing character and determines the printing of certain optional information.
procedure name	is the FMP file namr of a file containing REPORT commands stored as a procedure by the CREATE command or the RTE Editor in a procedure file.
body	consists of header statements, detail statements, edit statements, sort statements, group statements and total statements, as outlined in Table 3-2.

A REPORT command operates only on data entries whose record addresses are stored in the current select file. The user must enter a FIND command prior to using the REPORT command to specify which data entries are used by the REPORT command.

REPORT command output is printed on the list device. The line printer uses the first character of each output line as a carriage control character, and subsequently prints the character in column 2 of the output page in column 1 of the printed page. Care must be taken in assigning print positions for REPORT command output formats so that the first character of data in each report line is not lost.

REPORT ALL prints the entire data item and all elements of a compound item. The REPORT command form prints at most 132 characters of an item and only the first element of a compound item.

To use the REPORT NAME=procedure name form of the command, the user must first store a series of REPORT commands as a procedure in a file using a CREATE command or the RTE Editor. Once the procedure is stored, the user executes the procedure by typing REPORT NAME= procedure name . QUERY searches for the procedure and executes it. If the procedure named in the command does not exist, QUERY informs the user and prompts for another command by printing NEXT?. If an ASCII printing characters preceded by a comma follows the REPORT command, in the procedure file, QUERY prints a listing of the stored procedure prior to executing it on the terminal device. If the command does not contain the ASCII printing character, QUERY executes the procedure without listing it.

When a REPORT procedure is created using the CREATE command, QUERY does not check the procedure for legal format. When the procedure is requested with REPORT NAME=procedure name QUERY checks the procedure line by line. If an error is discovered, the offending line is printed on the terminal, followed by an error message. Report generation is terminated at this point. The user should correct the procedure file using the EXECUTE command and run the report again.

The REPORT ALL form of the command prints the data item names and the data item values for every entry specified in the select file. If the user enters REPORT ALL followed by a comma and an ASCII printing character, QUERY prints only the value of each data item without printing the data item name associated with that value.

To use the REPORT; body END; form of the command, the user types the word REPORT followed by a semi-colon. The body of the report should then follow. Each body statement ends with a semi-colon. When the user presses return, QUERY prompts for another line by typing a question mark. A sentence can occupy more than one line by pressing return prior to typing the semi-colon ending the sentence. After the user types a semi-colon (signifying the end of a sentence) QUERY checks the sentence for proper form.

The report body itself consists of the following sentence types: header, detail, edit, sort, group, and total. With these sentences the user controls each line of output printed by QUERY in response to a REPORT command.

Table 3-2. REPORT Statements

STATEMENT TYPE	FUNCTION
Header	Prints title, column headings and page numbers at the top of each report page.
Detail	Prints data item values in the column position specified.
Edit	Describes edit masks used to punctuate Group, Detail or Total fields.
Sort	Sorts data entries based upon the value of a specified data item.
Group	Prints a data item value or character string whenever the value of an appropriate sort item changes.
Total	Prints column count, average or totals for logical groups or entire report.

Designing A Report

Report formats vary according to their use. However, many reports assume the general format depicted in Figure 3-2. The HEADERS describe the report and are printed at the top of each page along with the page number. HEADERS are also used to describe the report columns.

The report body consists of DETAIL lines, GROUP titles, and TOTALS, along with other descriptive labels. Normally, each detail line displays information from a single data entry, although information from a single data entry can appear on more than one line. A DETAIL field can be edited to include commas, decimal points, dollar signs, and other punctuation characters.

DETAIL lines can be sorted and grouped according to the values of data items in the entry. For example, a sales report may list sales results by country, region, sales office, and finally by individual salesman within each office. A GROUP title can be printed whenever "sort field" changes value. For example, when the country changes, the name of the country could be displayed as a GROUP title. The title can be a series of characters or a data item value.

SUBTOTALS can be printed for logical groups (for example, for each sales office) and GRANDTOTALS for the entire report. These totals add, average or count the DETAIL fields in each column of the report. Like DETAIL and GROUP fields, TOTAL fields can be edited with puncuation characters.

```
                        TITLE OF REPORT                PAGE NO.

                HEADER      HEADER      HEADER

GROUP TITLE     DETAIL      DETAIL      DETAIL
                DETAIL      DETAIL      DETAIL
                DETAIL      DETAIL      DETAIL

               SUBTOTAL    SUBTOTAL    SUBTOTAL

GROUP TITLE     DETAIL      DETAIL      DETAIL
                DETAIL      DETAIL      DETAIL
                DETAIL      DETAIL      DETAIL

               SUBTOTAL    SUBTOTAL    SUBTOTAL

              GRANDTOTAL  GRANDTOTAL  GRANDTOTAL
```

Figure 3-2. General Report Format

In the report statement descriptions which follow, a report is created by adding one statement type at a time and showing how the added statements change the report. Figure 3-3 contains the final version of the report.

```
FIND FAIL.DATE IGT "781206" AND FAIL.FAILCD IS NOT " " END;
REPORT;
H1,"ASSEMBLY FAILURE REPORT",50;
H1,"PAGE",62;
H1,PAGENO,66,SPACE A3;
H2,"DATE      ASSEMBLY       PART      FAILURE    REFERENCE",67;
H3,"NUMBER       NUMBER      CODE     DESIGNATOR",67,SPACE A2;
S2,DATE;
S1,ASSY#;
S,PART#;
D1,PART#,44,E1;
D1,FAILCD,52;
D1,REFDES,64;
G1,ASSY#,32,SPACE B2,E2;
G2,DATE,18,E3;
T1,"FAILURE COUNT",54;
T1,ASSY#,64,SPACE B2,COUNT;
T2,"FAILURES THIS DAY",54;
T2,ASSY#,64,SPACE B2,COUNT;
TF,"TOTAL FAILURES",54;
TF,ASSY#,64,SPACE B2,COUNT;
E1,"XXXX-XXXX";
E2,"XXXXX-XXXXX";
E3,"XX/XX/XX";
END;
```

Figure 3-3. Sample QUERY Report File

Report Formatting

When defining a report, certain parameters can be used on the REPORT statements to aid in formatting. The definition of these parameters is the same no matter where they occur. These parameters are the print position, SPACE, space-control, SKIP, skip-control and Edit parameter.

Print position defines the ending column position of a character string or data item value that is to be printed. The first column on the left of the report printout is column 1. QUERY does not check for data item or character string overlap on printout. It is the user's responsibility to choose correct end print positions for values to be printed. The number chosen can range from 1 to 132, since each line in a report can contain a maximum of 132 characters.

SPACE skips lines on the report page before or after printing the lines. The user informs QUERY whether to space before or after and how many lines to skip by the use of the space-control parameter following the word SPACE.

Space-control is either A[number] or B[number]. "A" followed by an integer will cause the number of lines to be spaced after printing the line. "B" followed by an integer causes the number of lines to be spaced before printing the lines. The number is an integer from 1 to 5. If the number is not specified, QUERY spaces one line. To space before and after printing a line, use the space option twice. For example,

```
     H1,"PAGE",35,SPACE B2,SPACE A3;
```

SKIP indicates that QUERY is to skip to a new page before or after printing a detail, group or total line. The skip-control will indicate whether the skip is before or after printing a line. Normally, QUERY will print 54 lines per page before skipping to a new page.

Skip-control consists of the letter A, skip after printing this line, or B, skip before printing this line.

Edit parameter is either EZ or Enumber where number is an integer from 0 to 9. EZ indicates that QUERY is to suppress any leading zeroes in numeric data item values. Enumber corresponds to the number of an edit statement that is defined after the statement containing the edit parameter. One edit parameter can be specified on any one report statement. If no edit mask is specified and a data item value is negative, the negative sign will appear on the left side of the value. If an edit mask is specified, the negative indication will be suppressed for a negative value unless a CR or minus sign appears in the mask.

Whenever a character string is specified in a report command, it is bracketed by quotes. QUERY prints the character string (minus the quotes) ending in the column specified by print position. Any printing ASCII character (including blanks) can be used in the string. If quotes are to appear in the string, they must be represented as a pair of quotes. The pair is printed as only one quote in the character string.

HEADER Statements

The header statement is used to print heading information of the user's choice at the top of each page of the report. A maximum of five lines of header information can appear at the top of each report page. The format of the header statement is:

```
Hnumber,data type,print position[,SPACE space-control];
```

where:

number is an integer from 1 to 5 specifying on which header line (out of five possible lines) the information is to appear. Header information in a header statement labeled H1 appears in the first line, H2 appears in the second line, etc.

data type is either an ASCII character string bracketed by quotes or the word PAGENO.

If PAGENO appears in the header statement, QUERY prints the page number of the report in the position specified by print position. QUERY increments the page number automatically for each page printed.

print position, SPACE, space-control are as defined in the section on REPORT FORMATTING.

Example:

```
REPORT;
H1,"ASSEMBLY FAILURE REPORT",50;
H1,"PAGE",62;
H1,PAGENO,66,SPACE A3;
H2,"DATE      ASSEMBLY      PART      FAILURE    REFERENCE",67;
H3,"NUMBER       NUMBER      CODE     DESIGNATOR",67,SPACE A2;
END;
```

```
                              ASSEMBLY FAILURE REPORT        PAGE   1



       DATE       ASSEMBLY        PART       FAILURE     REFERENCE
                   NUMBER        NUMBER       CODE      DESIGNATOR
```

DETAIL Statements

The detail statement indicates which data items of a data entry specified in the select file are to be printed in the report. Data items can be printed on up to 10 different lines. QUERY prints only the values of data items which do appear in a detail statement. If the data item length exceeds the distance between column 1 and its print position, it will be truncated.

The format of the detail statement is:

```
     D[n],data type,print position[,SPACE space-control][,SKIP
       skip-control][,edit];
```

where:

n	is an integer from 1 to 9. Each number specifies a different line on which the data items are printed. If the number is omitted, the data type is printed on a group line when any control break occurs. If no group lines are specified, the unnumbered detail item is printed on a separate line above any numbered detail item lines. The lowest numbered statement is printed first and all others follow in numeric order. Detail statements with the same number are printed on the same line.
data type	is either an ASCII character string bracketed by quotes or the name of a data item contained in the data entries specified in the select file.
print position, SPACE, space-control, SKIP, skip-control and edit	are as defined in the section on REPORT FORMATTING.

Example:

```
REPORT;
H1,"ASSEMBLY FAILURE REPORT",50;
H1,"PAGE",62;
H1,PAGENO,66,SPACE A3;
H2,"DATE      ASSEMBLY      PART      FAILURE   REFERENCE",67;
H3,"NUMBER       NUMBER      CODE     DESIGNATOR",67,SPACE A2;
D1,PART#,44;
D1,FAILCD,52;
D1,REFDES,64;
END;
```

```
                    ASSEMBLY FAILURE REPORT        PAGE   1

  DATE       ASSEMBLY       PART       FAILURE     REFERENCE
              NUMBER       NUMBER        CODE      DESIGNATOR


                          59501111      102           K12
                          59501550      103           K12
                          59501550      201           L12
                          59501134      109           K41
                          59501134      203           J42
                          59501489      203           K12
                          59501489      105           J14
                          59501489      103           K9
                          59501111      103           K23
                          59501134      201           L17
                          59501550      307           H12
```

EDIT Statements

The edit statement is used for punctuating data item values printed in a report. The statement performs such functions as suppressing leading zeroes in numeric values, inserting characters such as dollar signs, dashes, commas, and decimal points, as well as masking characters to eliminate them from the printed output.

Up to ten edit statements, labeled from E0 to E9, can be used in a report. To edit output from a detail, group, or total statement, the user includes the label of the desired edit statement. QUERY uses the specifications of the labeled edit statement to edit the value printed by the detail, group or total statement. The same edit statement can be referenced by more than one statement in the same REPORT command. Each edit statement must be referenced at least once in the REPORT command.

The format of the edit statement is:

```
     Enumber,"edit mask";
```

where:

number	is an integer from 0 to 9, identifying the edit statement. No two edit masks may be identified by the same integer.
edit mask	consists of from 1 to 20 numberic edit characters or from 1 to 132 alphanumeric edit characters, bracketed by quotes.

Alphanumeric Editing

Alphanumeric edit masks consist of X's (used as place holders) and any other ASCII printing characters (used as insertion characters). QUERY examines the data item value in the detail, group or total statement and the edit mask specified in the referenced edit statement, starting with the rightmost character of each. If the character in the edit mask is X, QUERY prints a character from the data item value in the corresponding postion of the output field. If the character in the edit mask is any character other than an X, QUERY prints the edit mask character in the corresponding position of the output field.

If there are fewer X's in the edit mask than there are characters in the data item value, the leftmost characters of the data item value are not printed. If there are more X's in the edit mask than there are characters in the data item value, QUERY prints asterisks in the extra columns. All insertion characters indicated in the edit mask are printed in the output field.

Figure 3-4 shows the results of printing data item value characters strings using edit masks.

Data Item Value	Edit Mask	Printed Result
ABCD	"X-X-X-X"	A-B-C-D
ABCD	"XX"	CD
ABCD	"XXXXX"	*ABCD
ABCD	"X//X-X-X-X"	*//A-B-C-D

Figure 3-4. Alphanumeric Edit Examples

Numeric Editing

A numeric edit mask consists of the placement holders (9,Z,*,$), sign character (CR,-) and any other ASCII printing characters used as insertion characters. The numeric edit mask edits integer numeric values consisting of the digits 0-9.

The edit mask consists of up to 20 characters in the combinations

The length of the edit mask determines the length of the output field printed by QUERY. If the number of digits of the data item value is greater than the number of place holders (9,Z,*,$) in the edit mask, QUERY prints all asterisks in the output field. However, if the number of digits is less than the number of place holders in the edit mask, the value is left-filled with spaces.

Real numbers are reported but they may not be edited. They must have a 13 column field. Fixed numbers will be reported whenever the field will permit, otherwise the real number will be reported using exponential notation. The format will be in a FORTRAN G13.5 format.

Table 3-3. Numeric Edit Mask Combinations

Combinations	Examples
9's only	"9999"
Z's only	"ZZZZZ"
Asterisks only	"*******"
Dollar signs only	"$$$$$$"
Sign characters and 9's	"9999CR" "9999-"
Sign characters, 9's, and either Z's, asterisks or dollar signs	"$$$999CR" "ZZ9999-" "****9999999"
9's and either Z's, asterisks or dollar signs	"$$$99999" "ZZZZ999999" "***99999"
Any of the above combinations including insertion characters such as commas, one decimal point, slashes, etc., located anywhere in the edit mask, except to the right of sign characters.	"$$999,999.99-" "999-99-9999" "99/99/99" "$$9999.99CR"

Only one decimal point may appear in any numeric edit mask. If a minus sign appears in the mask in any position other than the rightmost character of the mask, the minus is treated as an insertion character.

Each of the place holders and sign characters serves a special purpose in editing data item values. The characters and their meanings are specified in Table 3-4.

Table 3-4. Edit Mask Characters

Character	Explanation
9	Each 9 in the edit mask is replaced with a decimal digit from the data item value in the corresponding position of the output field.
Z	Z is a zero suppression place holder. A Z in the edit mask is replaced with a decimal digit from the data item value in the corresponding position of the output field. If the data item value digit under consideration is a zero appearing to the left of the leftmost significant digit, QUERY inserts a blank in the output field, and all other zeroes to the left of the significant digit are replaced by a blank in the output field.
*	* is an asterisk place holder. An * in the edit mask acts just like a Z with the exception that leading zeroes in the data item value are replaced with asterisks in the output field.
$	$ is the dollar sign place holder. A $ in the edit mask acts just like a Z, except that the first zero in the data item value to the left of the leftmost significant digit is replaced with the dollar sign in the output field. All zeroes to the left of the first leading zero are replaced with blanks in the output field.
CR	CR is a sign character and always appears in the two rightmost positions of the edit mask. If the data item value is negative, QUERY prints the two characters CR in the first two rightmost positions of the output field. If the data item value is positive, QUERY prints two blank characters in place of the CR. No characters from the data item value are ever placed in the first two rightmost positions of the output field. If CR is put in the two leftmost characters, QUERY does not print the CR or the minus sign.
-	- is a sign character and acts the same as the CR character. If the data item value is negative, QUERY prints the minus sign (-) in the rightmost position of the output field. If the data item value is positive, QUERY prints a blank in place of the minus sign. No character from the data item value is ever placed in the rightmost position of the output field.

Insertion Characters	Insertion characters consists of any ASCII printing characters not previously mentioned. Insertion characters are printed in the output field in the same position they appear in the edit mask. Any insertion character appearing in the edit mask to the left of the leftmost significant digit of the data item value is replaced with blanks or an asterisk, depending on which zero suppression character is specified in the edit mask. Only one decimal point can appear in an edit mask.

Figure 3-5 shows the results of printing numeric data item values using numeric edit masks.

Data Item Value	Edit Mask	Edited Result
0059	"$$$,999"	$059
1024	"ZZZ,ZZZ"	1,024
010555	"$$,$$$.99CR"	$105.55
-010555	"$$,$$$.99CR"	$105.55CR
-010555	"$$,$$$.99-"	$105.55-
010555	"$$,$$$.99-"	$105.55
15039250	"$,$$$,$$$.99CR"	$150,392.50
-1399	"*,***.99CR"	***13.99CR
044240474	"999-99-9999"	044-24-0474
-2145	"$,$$$.99"	$21.45

Figure 3-5. Numeric Edit Examples

Example:

```
REPORT;
H1,"ASSEMBLY FAILURE REPORT",50;
H1,"PAGE",62;
H1,PAGENO,66,SPACE A3;
H2,"DATE      ASSEMBLY       PART      FAILURE    REFERENCE",67;
H3,"NUMBER        NUMBER      CODE     DESIGNATOR",67,SPACE A2;
D1,PART#,44,E1;
D1,FAILCD,52;
D1,REFDES,64;
E1,"XXXX-XXXX";
END;
```

```
                     ASSEMBLY FAILURE REPORT        PAGE   1


     DATE        ASSEMBLY         PART        FAILURE     REFERENCE
                  NUMBER         NUMBER        CODE      DESIGNATOR


                              5950-1111        102           K12
                              5950-1550        103           K12
                              5950-1550        201           L12
                              5950-1134        109           K41
                              5950-1134        203           J42
                              5950-1489        203           K12
                              5950-1489        105           J14
                              5950-1489        103           K9
                              5950-1111        103           K23
                              5950-1134        201           L17
                              5950-1550        307           H12
```

SORT Statements

The sort statement serves two purposes in the report.

- Specifies data items (called sort items) whose values are used by QUERY to sort data entries before they are printed in the report.
- Defines control break levels for use by group and total statements in the REPORT command.

The format of the sort statement is

```
S[level],data item name;
```

where:

level	is an integer from 1 to 5
data item name	is the name of a data item contained in the data entries specified in the select file.

Each sort item specified in any level is used to create a sort key. The sort key may not be longer than 80 bytes. The total length of the sort key is calculated by summing the length of all the specified sort items. An I1 item requires 2 bytes, an R2 item requires 4 bytes and an Xn item requires n bytes. When a compound item is specified as part of a sort key, the total length of the item must be used to calculate the length of the key. However, only the first element is actually used in the sort.

Under normal conditions, data entries are listed in a report in the order in which they occurred in the select file as they were discovered by the FIND command. These data entries are not arranged in any alphabetic order or numeric sequence. The sort statement defines a data item belonging to the data entries in the select file whose value is used to sort data entries into order. The data item defined in the sort statement is called a sort item.

The sort statement:

```
     S1,LNAME;
```

sorts the entries selected into the order specified by the value of LNAME.

Data Entries After FIND			Data Entries After Sort Executed		
LNAME	FNAME	AGE	LNAME	FNAME	AGE
WHITE	DANA	26	BROWN	JILL	32
BROWN	JILL	32	BROWN	CHRIS	17
GREEN	DALE	49	BROWN	DAN	39
WHITE	LAURA	81	GREEN	DALE	49
BROWN	CHRIS	17	GREEN	SALLY	28
GREEN	SALLY	28	GREEN	BILL	45
GREEN	BILL	45	WHITE	DANA	26
BROWN	DAN	39	WHITE	LAURA	81
WHITE	WILL	22	WHITE	WILL	22

It is possible to sort on many different data items. The higher-numbered sort statement identifies the major sort field, while the lower-numbered sort statement identifies the minor sort field. The minor sort arranges entries in the order specified, keeping all major sort items with identical values together. For example, if the statement illustrated above appeared in a report with another statement:

```
     S1,FNAME;
     S2,LNAME;
```

the result would be as follows:

```
                    LNAME                           FNAME     AGE

level 2 (major)     BROWN   level 1 (minor)         CHRIS     17
control break       BROWN   control breaks          DAN       39
                    BROWN                           JILL      32
level 2 (major)     GREEN                           BILL      45
control break       GREEN                           DALE      49
                    GREEN                           SALLY     28
level 2 (major)     WHITE                           DANA      26
control break       WHITE                           LAURA     81
                    WHITE                           WILL      22
```

Control Breaks

A control break occurs during the printing of a report whenever the current entry's value for a data item defined in a numbered sort statement is different from the last entry's value. When the first entry is printed, a control break occurs since the data item value changes from null (no value) to the first value. Totals are not printed when the first control break occurs.

In the example above, a control break occurs when the value of LNAME becomes BROWN and when it changes to GREEN and again when it changes to WHITE. This is known as a level 2 control break because the data item name LNAME appears in a sort statement labeled S2. The level 1 control break is associated with the data item FNAME and sort statement labeled S1.

A control break occurs for all lower levels whenever a higher level control break occurs. This means that whenever a control break occurs for level 2 (LNAME) a control break occurs for level 1 (FNAME) by definition.

A group or total statement prints only when a control break occurs that is at the same level as the group or total statement. This means that a total statement labeled T2 prints only when a level 2 control break occurs. Consult the descriptions of group and total statements below for explanations of their functions.

Sort statements without a level (i.e., no number) are used to sort entries but do not define control breaks for use by group or total statements.

Major to Minor Sort Fields

Numbered and unnumbered sort statements can appear in the same REPORT command. The order in which unnumbered sort statements appear in the report body is significant. The first unnumbered statement defines the most minor sort field, while the last unnumbered statement defines the most major sort field. QUERY defines sort fields in the following order from most major to most minor:

```
Most Major                                             Most Minor

S5 S4 S3 S2 S1 S(last in report body)...S(first in report body)
```

For example, the sort statements below define the order as shown:

Statements	Order		
S2,OFFICE	MONTH	major	(S3)
S,PARTNO	OFFICE		(S2)
S1,SLSMAN	SLSMAN		(S1)
S,QUAN	QUAN		(S last)
S3,MONTH	PARTNO		(S first)

Example:

```
REPORT;
H1,"ASSEMBLY FAILURE REPORT",50;
H1,"PAGE",62;
H1,PAGENO,66,SPACE A3;
H2,"DATE      ASSEMBLY        PART       FAILURE    REFERENCE",67;
H3,"NUMBER       NUMBER       CODE      DESIGNATOR",67,SPACE A2;
S2,DATE;
S1,ASSY#;
S,PART#;
D1,PART#,44,E1;
D1,FAILCD,52;
D1,REFDES,64;
E1,"XXXX-XXXX";
END;
```

```
                        ASSEMBLY FAILURE REPORT        PAGE   1


      DATE      ASSEMBLY       PART       FAILURE    REFERENCE
                 NUMBER       NUMBER       CODE      DESIGNATOR


                             5950-1111     102          K12
                             5950-1550     103          K12
                             5950-1550     201          L12
                             5950-1134     109          K41
                             5950-1134     203          J42
                             5950-1489     203          K12
                             5950-1489     105          J14
                             5950-1489     103          K9
                             5950-1111     103          K23
                             5950-1134     201          L17
                             5950-1550     307          H12
```

GROUP Statements

The group statement is used to print the value of data items or character strings whenever a control break occurs at a specific control level. Each control break level is defined by a sort statement labeled with an integer from 1 to 5. A group statement is assigned to a control break level by using the same level number as the defining sort statement. Whenever a break occurs during the printing of a report, QUERY prints a data item value or literal character string as defined by the group statement corresponding to the control break level.

A breakpoint occurs not only when the sort item defined in the sort statement changes value, but also at the very beginning of the report, where the sort item value changes from empty to some non-empty value. When a breakpoint occurs for a major sort item, all minor sort items (defined by sort statements with levels less than the major sort item control level) are also at breakpoint, by definition. If major and minor group statements are defined in the REPORT command, QUERY prints a data item value or characters string for every control break level equal to or below the major sort item control break level.

The format of a group statement is:

```
     Glevel,data type,print position[,SPACE space-control][,SKIP
       skip-control][,edit];
```

where:

level   is an integer from 1 to 5, corresponding to the level appearing in a sort statement.

data type is either an ASCII character string bracketed by quotes, or the name of a data item.

print position, SPACE, space-control, SKIP, skip-control and edit are as defined in the section on REPORT FORMATTING.

If the data type is an ASCII character string, QUERY prints the string (minus the quotes) whenever the group statement is executed. If the character string is to contain quotes, they must appear in the string as a pair of quotes to distinguish them from the quotes bracketing the string. QUERY prints the pair of quotes as one quote. If the data type is a data item name, QUERY prints the new value of the data item which defines a control breakpoint.

Example:

```
REPORT;
H1,"ASSEMBLY FAILURE REPORT",50;
H1,"PAGE",62;
H1,PAGENO,66,SPACE A3;
H2,"DATE      ASSEMBLY       PART       FAILURE    REFERENCE",67;
H3,"NUMBER        NUMBER     CODE      DESIGNATOR",67,SPACE A2;
S2,DATE;
S1,ASSY#;
S,PART#;
D1,PART#,44,E1;
D1,FAILCD,52;
D1,REFDES,64;
G1,ASSY#,32,SPACE B2,E2;
G2,DATE,18,E3;
E1,"XXXX-XXXX";
E2,"XXXXX-XXXXX";
E3,"XX/XX/XX";
END;
```

```
                    ASSEMBLY FAILURE REPORT        PAGE   1


   DATE       ASSEMBLY       PART      FAILURE   REFERENCE
               NUMBER       NUMBER      CODE    DESIGNATOR



 78/12/07   92060-16001
                          5950-1111     102          K12
                          5950-1550     103          K12
                          5950-1550     201          L12

            92060-18001
                          5950-1134     109          K41
                          5950-1134     203          J42


            92060-60001
                          5950-1489     203          K12


            92060-80001
                          5950-1489     105          J14

 78/12/28   92060-16001
                          5950-1489     103          K9


            92060-18001
                          5950-1111     103          K23


            92060-80001
                          5950-1134     201          L17
                          5950-1550     307          H12
```

Notice that the two data items mentioned in the detail statements do not print on the same line as the group statements. This is because the detail statements are numbered. Unnumbered detail statements print on the same line as a group statement whenever a control break occurs.

TOTAL Statements

The total statement prints a data item value, character string, the total of a group of detail item values, the average of a group of detail values, and/or the number of detail item values printed, whenever a control break occurs. Each control break level is defined by a sort statement labeled with an integer from 1 to 5. A total statement is assigned to a control break level by using the same level number as the defining sort statement. Whenever a control break occurs during the printing of a report, QUERY prints the information as specified in the total statement corresponding to the control break level. There is a limit of 5 different items being counted, added, or averaged in any report.

A breakpoint occurs not only when the sort item defined in the sort statement changes value, but also at the end of the report. Total statements labeled with the letter F (instead of an integer from 1 to 5) print information after the last detail lines are printed in the report. At the end of the report (or at any level control break) each differently numbered total is printed on a separate line.

The format of the total statement is:

```
     Tlevel,data type,print position[,SPACE space-control][,SKIP
                                   ,AVERAGE
       skip-control][,edit]        ,COUNT   ;
                                   ,ADD
```

where:

level — is an integer from 1 to 5 corresponding to the level appearing in a sort statement or the letter F. The letter F indicates that information is printed only at the end of the report after the last detail line in the report is printed.

data type — is either an ASCII character string bracketed by quotes or the name of a data item.

print position, SPACE, space-control, SKIP, skip-control, and edit are as defined in the section on REPORT FORMATTING.

ADD — instructs QUERY to add the values specified in data type. Accumulators are set up for each control level and, if the user specifies F as the level, for the final level. The capacity is 10 digits.

AVERAGE — instructs QUERY to average the values of the data item specified in data type.

COUNT instructs QUERY to print the number of data item values printed after the last control break and before the current control break.

If data type is an ASCII string, QUERY prints the string (minus the quotes) whenever the total statement is executed. If the character string is to contain quotes, they must appear in the string as a pair of quotes to distinguish them from the quotes bracketing the string. QUERY prints the pair of quotes as one quote. If the data item name is used, QUERY prints the current value of the data item at the time of the control break. This will be the new value of the data item rather than the old value.

If the ADD option is specified, QUERY adds the value of the specified data item until a control break occurs. The accumulated value is printed in the report and the accumulator is set to zero for the next control group. The data item to be added must be an integer, real number, or numeric ASCII string of 20 digits or less in length (DECAR format).

If the AVERAGE option was specified, the data item values are added and then averaged for each group of data item values prior to a control break. When a control break occurs, QUERY prints the average of the data item values for the data entries printed after the last control break and prior to the current control break. QUERY then sets the accumulator to zero and starts a new average the next control group. The data item to be averaged must be an integer, real number or valid (DECAR format) ASCII string.

When count is specified, at the occurrence of a control break, QUERY prints the count of detail item values, then sets the accumulator to zero for the next control group.

If a total statement labeled with the letter F is specified, a total line is printed after the last detail line of the report. If this TF line contains an ADD, AVERAGE or COUNT option, an accumulation will have been made for each entry in the select file. A data item can appear in a TF line even if it has not appeared in any previous total statement.

One total statement must appear in the REPORT command for each ADD, COUNT or AVERAGE option desired, as a total statement can specify only one of the three at a time. The desired information is printed on the same line in the print position specified in the total statement. It is the user's responsibility when using more than one of the three options to insure that the information printed on the same line does not overlap.

If the TOTALLED value overflows or underflows, the ADDED or AVERAGED field is filled with asterisks.

```
REPORT;
H1,"ASSEMBLY FAILURE REPORT",50;
H1,"PAGE",62;
H1,PAGENO,66,SPACE A3;
H2,"DATE      ASSEMBLY       PART      FAILURE    REFERENCE",67;
H3,"NUMBER       NUMBER      CODE     DESIGNATOR",67,SPACE A2;
S2,DATE;
S1,ASSY#;
S,PART#;
D1,PART#,44,E1;
D1,FAILCD,52;
D1,REFDES,64;
G1,ASSY#,32,SPACE B2,E2;
G2,DATE,18,E3;
T1,"FAILURE COUNT",54;
T1,ASSY#,64,SPACE B2,COUNT;
T2,"FAILURES THIS DAY",54;
T2,ASSY#,64,SPACE B2,COUNT;
TF,"TOTAL FAILURES",54;
TF,ASSY#,64,SPACE B2,COUNT;
E1,"XXXX-XXXX";
E2,"XXXXX-XXXXX";
E3,"XX/XX/XX";
END;
```

And our final report looks like this:

```
                    ASSEMBLY FAILURE REPORT        PAGE    1


   DATE        ASSEMBLY        PART       FAILURE    REFERENCE
                NUMBER        NUMBER       CODE     DESIGNATOR


 78/12/07    92060-16001
                             5950-1111     102          K12
                             5950-1550     103          K12
                             5950-1550     201          L12

                                  FAILURE COUNT       00003


             92060-18001
                             5950-1134     109          K41
                             5950-1134     203          J42

                                  FAILURE COUNT       00002


             92060-60001
                             5950-1489     203          K12

                                  FAILURE COUNT       00001


             92060-80001
                             5950-1489     105          J14
                                  FAILURE COUNT       00001


                              FAILURES THIS DAY       00007

 78/12/28    92060-16001
                             5950-1489     103          K9

                                  FAILURE COUNT       00001


             92060-18001
                             5950-1111     103          K23
                ASSEMBLY FAILURE REPORT        PAGE   2

DATE      ASSEMBLY       PART       FAILURE    REFERENCE
           NUMBER       NUMBER       CODE     DESIGNATOR


                               FAILURE COUNT     00001


         92060-80001
                     5950-1134     201           L17
                     5950-1550     307           H12

                               FAILURE COUNT     00003

                           FAILURES THIS DAY     00004

                              TOTAL FAILURES     00011
```

Storing And Maintaining Procedures

Procedures provide a convenient way of storing QUERY commands for repeated use without having to retype them. Three types of procedures are used within QUERY.

- FIND procedures retrieve data entries from the database.
- REPORT procedures print reports about data retrieved from the database by the FIND command.
- UPDATE procedures add, delete or modify data entries in the database located by the FIND command.

Procedures are stored in an FMP disc file. If running QUERY remotely, the procedure file must be local to the node at which QUERY is executing. The QUERY user can manipulate procedures using File [illegible] Manager commands or the following QUERY commands.

- CREATE command defines the procedure and stores it in a disc file.
- DISPLAY command lists a procedure stored in a disc file.
- EXECUTE command allows execution of the RTE Editor to modify any procedure on the disc.
- DESTROY command deletes any procedure on the disc.

The format and use of QUERY procedures are detailed under the individual descriptions of the FIND, REPORT and UPDATE commands.

CREATE Command

The CREATE command defines FIND, REPORT or UPDATE procedures and stores them on the disc. The format of the command is:

```
     CREATE NAME=procedure name;
```

where:

procedure name	is the name of the procedure given by the user. A procedure name consists of an FMP file namr.

When a user types a CREATE command, QUERY types a question mark to prompt the user to enter the procedure. The user enters the procedure line by line. If the user fails to end a line with a semi-colon or a zero length record, QUERY will concatenate the line with the line following it before placing them in the procedure file.

The user enters the procedure statements according to the format described under the information for the individual command. QUERY continues to prompt for lines until the user types "END;" as the last four characters in a line. A space after END; will cause the line to be interpreted as other than the end of the procedure file.

After the user types END;, QUERY loads the procedure on the disc and returns to NEXT?. QUERY does not check the format of the procedure entered. Format checking for procedures occurs at the time the procedure is executed.

```
    NEXT?CREATE NAME=QAPRC:100:43;
    ? UPDATE A,FAIL;END;
    NEXT?
```

To execute the procedure, the user types:

```
     UPDATE NAME=QAPRC:100:43;
```

QUERY first lists the procedure on the log device and then executes the procedure. The user types in the values following the question mark after each data item name, as he would normally do when doing an UPDATE ADD.

The following is an example of a FIND command stored as a procedure on the disc through a CREATE command.

```
    NEXT?CREATE NAME=PRC2:100:43;
    ?FIND ASSY# IS "9206018001" END;
    NEXT?
```

To execute the above procedure, the user types:

```
     FIND NAME=PRC2:100:43;
```

After searching for data entries in the current data set with the characteristics defined in the FIND procedure, QUERY reports how many data entries were discovered with the specified characteristics.

A REPORT file can also be stored as a procedure using the CREATE command. The report statements are not checked for syntax errors at the time the file is created, but rather at the time the report procedure is executed.

DISPLAY Command

The DISPLAY command lists a procedure stored on the disc.

The format of the command is:

```
     DISPLAY NAME=procedure name;
```

where:

procedure name is the FMP file namr of a procedure stored on the disc.

When the user types DISPLAY NAME=procedure name;, QUERY prints the procedure specified on the list device. If the procedure does not exist on the cartridge specified (or on any cartridge, if no cartridge reference number is specified), QUERY informs the user and prompts for another commmand.

EXECUTE Command

The EXECUTE command provides access to the RTE Editor for the purpose of editing lines in a procedure file. EXECUTE also allows for the execution of any other program.

The format of the command is:

```
     EXECUTE[=program name];
```

where:

program name is the name of any program.

The default is EDITR. When QUERY is executing remotely, the EDITR will be scheduled using remote parameters. When the Editor with remote capabilities is not available at the node, the local Editor will be executed. If a program other than EDITR is specified, it will be scheduled with the same parameters as were used to schedule QUERY. The user should be sure to have the remote Editor available if EDITR is to be executed from remote QUERY. By the same token, any other program to be executed from remote QUERY that expects interactive input should be a remote, rather than a local program.

DESTROY Command

The DESTROY command deletes files stored on a disc. It can be used to remove procedure files from disc. The command format is:

```
     DESTROY NAME=procedure name;
```

where:

procedure name is the FMP file namr of a procedure stored on disc.

When the user enters the DESTROY command and a file exists with the FMP file namr given in the command, QUERY purges the file. If the procedure does not exist, QUERY informs the user and prompts for another command. QUERY does not verify that the file named in a DESTROY command does contain a procedure, so the user must use this command with caution.

Terminating QUERY

The EXIT command terminates QUERY execution and returns control to the operating system.` The format of the command is:

```
     EXIT;
```

EXIT should be used only in response to NEXT? typed by QUERY. The user should never terminate QUERY operation in any other way than with the EXIT command. Using the system break and then "offing" the program will leave files open and could possibly cause some damage to the database.

Utility Commands

The XEQ, FORM, HELP and LIST commands supply utility functions to QUERY users as follows:

- The XEQ command allows the user to enter several commands from a command file.
- The FORM command documents the database structure, including names of the data sets, names and length of data items, and the identification of key items defined as links between data sets.
- The HELP command provides a reference to the syntax and function of the QUERY commands. The user can gain assistance in the use of QUERY commands from the QUERY terminal.
- The LIST command allows the user to change the logical list device during a QUERY session.

XEQ Command

The XEQ command allows a user to enter several QUERY commands from a command file. When the XEQ command is entered, the specified file is read and the commands executed until an end-of-file or another XEQ command is encountered. Any command parameters are also read from the batch file. When an end-of-file is reached, control is returned to the original input device. When an error occurs, the error is reported to the log device and control is returned to the original input file or device. The original input file or device is that file or device specified in the RU,QUERY command.

When an XEQ command is encountered within a command file, the first command file is closed and the second one is opened. The old XEQ command file is never reopened.

The format of the command is:

```
     XEQ=file name;
```

where:

file name is the FMP file namr of an ASCII file containing commands and parameters. The command parameters must follow the command in the file. Command parameters are any input expected by the command.

An example of a command file is:

```
     DATA-BASE=QA:100:43;
     ADMIN;
     1;
     SELECT-FILE=ASEL::43;
     FIND PROJ.PROJ# IS "940071"END;
     UPDATE R;
     OPDAT="790416";
     CLDAT="791010";
```

If the FMP file namr of the above file is TRFL:100:43, then the following statement would transfer control to that file:

```
     XEQ=TRFL:100:43;
```

Notice that there is a record in the command file for each line of input expected by QUERY. If executing commands via XEQ that expect input, the command file must include that input.

Upon reaching the end of file, QUERY will return to interactive mode by printing NEXT?, if the input device is interactive. If not, QUERY will resume executing commands in the original input file.

If an error occurs while executing an XEQ command, an error message reported. If XEQ was entered interactively, QUERY will return the

FORM Command

The FORM command prints a description of the database currently open. The format of the command is:

```
FORM;
```

The FORM command provides a description of each data set in a database on the list device. QUERY lists the data set name, type (manual master, automatic master, detail) and capacity, along with a list of the elements of each data set and the length in words and type of each element. QUERY also identifies key item and whether there is write access for the items.

Information is displayed only for those items which have a read level less than or equal to that at which the database was opened. All other items are not displayed.

The following is a sample printout resulting from a FORM command:

```
                    * * * * IMAGE/1000 SCHEMA * * * *
                           (USING LEVEL 15  )

DATA SET - ASMBY ,M  CAPACITY = 0000000053

  ITEM NAME    TYPE    LENGTH    KEY ITEM    SORT ITEM    # ELEMTS    WRT ACCE

    ASSY#       X       0010       YES                       001         YES
    FAIL#       I       0001                                 001         YES

DATA SET - PART  ,M  CAPACITY = 0000000053

  ITEM NAME    TYPE    LENGTH    KEY ITEM    SORT ITEM    # ELEMTS    WRT ACCE

    PART#       X       0008       YES                       001         YES

DATA SET - CODE  ,A  CAPACITY = 0000000053

  ITEM NAME    TYPE    LENGTH    KEY ITEM    SORT ITEM    # ELEMTS    WRT ACCE

    FAILCD      X       0004       YES                       001         YES

DATA SET - DATEF ,A  CAPACITY = 0000000053
```

```
   ITEM NAME    TYPE    LENGTH    KEY ITEM    SORT ITEM    # ELEMTS    WRT ACCE

     DATE         X      0006       YES                      001         YES

DATA SET - REF    ,A   CAPACITY = 0000000053

   ITEM NAME    TYPE    LENGTH    KEY ITEM    SORT ITEM    # ELEMTS    WRT ACCE

     REFDES       X      0004       YES                      001         YES

DATA SET - EMPL   ,A   CAPACITY = 0000000101

   ITEM NAME    TYPE    LENGTH    KEY ITEM    SORT ITEM    # ELEMTS    WRT ACCE

     EMP#         X      0006       YES                      001         YES

DATA SET - PROJ   ,M   CAPACITY = 0000000053

   ITEM NAME    TYPE    LENGTH    KEY ITEM    SORT ITEM    # ELEMTS    WRT ACCE

     PROJ#        X      0006       YES                      001         YES
     DEPT#        X      0004                                001         YES
     OPDAT        X      0006                                001         YES
     CLDAT        X      0006                                001         YES

DATA SET - FAIL   ,D   CAPACITY = 0000000101

   ITEM NAME    TYPE    LENGTH    KEY ITEM    SORT ITEM    # ELEMTS    WRT ACCE

     ASSY#        X      0010       YES                      001         YES
     DATE         X      0006       YES                      001         YES
     PART#        X      0008       YES                      001         YES
     FAILCD       X      0004       YES                      001         YES
     REFDES       X      0004       YES                      001         YES
     PNTS         X      0001                                008         YES
     EMP#         X      0006       YES                      001         YES
     SUP#         X      0006       YES                      001         YES

DATA SET - TIME   ,D   CAPACITY = 0000000101

   ITEM NAME    TYPE    LENGTH    KEY ITEM    SORT ITEM    # ELEMTS    WRT ACCE

     EMP#         X      0006       YES        PROJ#         001         YES
     PROJ#        X      0006       YES        DATE          001         YES
     DATE         X      0006       YES        HOURS         001         YES
     HOURS        R      0002                                001         YES
```

HELP Command

The HELP command prints messages to the user on the list device explaining the purpose and form of any QUERY command. The format of the command is:

```
     HELP[,command][ ,FUNCTION][ ,SYNTAX][ ,OPERANDS];
```

where:

command is one of the following commands: CREATE DATA-BASE

Chapter 4
Host Language Access

After the database is designed and the root file and data sets have been created, application programs can be written that will be used to enter and manipulate the data. Programs written in FORTRAN, PASCAL, Assembly Language or BASIC gain access to the database through calls to IMAGE library subroutines (note that BASIC is not available to 92073A users). Ten subroutines are provided. The subroutines and their functions are:

DBOPN	Initiates access to the database and defines the user's level of access.
DBINF	Provides information about the organization and components of the database being accessed.
DBFND	Locates the beginning of a data chain in preparation for access to entries in the chain.
DBGET	Reads data items.
DBUPD	Modifies the values of data items in existing data records.
DBPUT	Adds new data records.
DBDEL	Deletes existing data records.
DBLCK	Locks a database temporarily to provide exclusive access.
DBUNL	Unlocks a database previously locked by a call to DBLCK.
DBCLS	Closes the database files.

Specific information about procedure calls, parameters and status information is given later in this chapter. A general description of the functions available will be found in this chapter, followed by a discussion of the parameters used when invoking the functions. Following these general descriptions are specific calling sequences, ordered alphabetically, for easy reference. In each description of a call, references to further information about text or parameter discussion, as well references to examples, will be found. Actual examples are presented in Chapter 5 Note that before the application programs can be executed, the database must be created using the IMAGE Schema Processor described in Chapter 2.

Opening The Database

Before a user can gain access to the data, he must open the database with a call to the DBOPN routine. In opening a database, DBOPN establishes an access path between the database and the program by:

- verifying the right to use the database under the security provisions provided by the RTE file system and the IMAGE level code word scheme.

- determining that the access mode requested in opening the data base is compatible with the access modes of other users currently using the database.

- calculating whether there is enough space between the end of the user's program and the end of the partition to hold the Run Table.

- opening the root file and constructing a Run Table to be used by all other IMAGE subroutines when they are executed. The Run Table contains both static and dynamic information about the database. The information in the Run Table is initially derived from the root file but is modified slightly according to the access level specified when DBOPN is called. The root file remains open until the database is closed.

Run Table

A Run Table is a table of database relative information, both static and dynamic, used by the IMAGE subroutines in providing and controlling access to the database. The Run Table is created when the database is opened and is destroyed when the database is closed. During its existence, it is kept in the user's partition in the space from the end of the user's program to the end of the partition. If the program uses the EMA capability of the RTE Operating System, the Run Table is located between the end of the user's program and the beginning of the MSEG. For more information about the Run Table, see Chapter 7.

DBCOP

DBCOP (Database Coordinating Program) is a program which must be available for scheduling whenever an IMAGE program is running. Attempting a call to an IMAGE subroutine without having DBCOP available will result in a status code of 140 being returned. For more information about DBCOP, see Chapter 7.

Segmented Programs And IMAGE

IMAGE keeps track of certain primary information in a routine called DBBUF. The Database Pointer Table, the DCB Pointer Table and the Record Pointer are kept in DBBUF. It is important that DBBUF be available to all segments, if running IMAGE from a segmented program. For this reason, all IMAGE segmented programs should include a dummy call to DBBUF in the main segment. Failure to include DBBUF in the main segment will result in either an IMAGE error 103 or 160.

Open Modes

There are currently three different open modes available. Each mode determines the type of operation that can be perform on the database as well as whether other users may access the database concurrently. The three open modes and the access allowed with each are:

MODE	ACCESS	SHARED/EXCLUSIVE
1	Read/Write	Shared
3	Read/Write	Exclusive
8	Read Only	Shared

Modes 2 and 4 through 7 are currently unused.

Obtaining Information About The Database Structure

The DBINF library subroutine allows the user to programmatically acquire information about the database. It provides information about data items, data sets, data paths, or current access paths. The information returned is restricted by the access level established when the database is opened.

Any data items, data sets, or paths of the database inaccessible in that access level are considered to be non-existent. For example, if the level of the user is lower than the read level of an item, that item will not be reported by DBINF.

The information that can be obtained through separate calls to DBINF is summarized below.

In relation to data items, DBINF can be used:

- to determine whether the access level established when the database was opened allows a specified data item value to be read only or read and written.
- to get a description of a data item including the data item name, type, and length. This information corresponds to that which is specified in the item part of the schema.
- to determine the number of items in the database available to the user and to get a list of numbers identifying those items. The numbers indicate the position of each data item in the item part of the schema. The type of access can also be determined.
- to determine the number of items in a particular data set available to the user and get a list of those item numbers and the type of access available for each one.

In relation to data sets, DBINF can be used:

- to determine whether the user can only read entries or can add, delete and update entries.
- to get a data set description including the data set name, type and length in words for data entries in the set, number of entries in the set, and the capacity.
- to determine the number of data sets the user can access and the type of access allowed.
- to determine in which data sets a particular data item is available to the user. The number of data sets, a list of data set numbers, and the type of access available for each set is returned.

In relation to data paths, DBINF can be used:

- to get information about the paths associated with a particular data set including the number of paths.
- to determine the key item number of a master data set.

In relation to current access paths, DBINF can be used:

- to obtain the current chain information for the specified data set.
- to restore previously obtained chain information as current chain information.

Special Uses Of DBINF

DBINF can be used to obtain information such as item length and using this information, item offset can be calculated. Using DBINF in this manner will make a program more flexible and less sensitive to changes in the structure of the database.

If the application is key item dependent, the current access path calls to DBINF can be used as place holders on chain reads. This allows a program to chain down various paths in any one data set without having to start each chain read at the beginning of the access path.

Reading The Data

When data is read from the database the user specifies which data set and which entry in that data set contains the information desired. The user will have access to those data items in the entry to which he is entitled based on the access level supplied when the database was opened. The entire entry may be read (if so entitled) or specific data items from the entry. The items to be read and the array where the values should be stored are specified by the user.

To understand the various ways in which the data entry to be read can be selected, it is important to know a little about the data set structure. Each data set consists of one disc file and each data entry is a logical record in that file. Each entry is identified by the relative record number in which it is stored. The first record in the data set is record number 1 and the last is record number n where n is the capacity of the data set.

At any given time, a record may or may not contain an entry. IMAGE maintains internal information indicating which records of a data set contain entries and which do not.

Current Path

IMAGE maintains a current path for each detail data set. The path is established by the DBFND call. Each time an entry is read through a chained read, IMAGE saves the entry's backward and forward chain pointers for the current path.

Reading Methods

The methods for requesting a data entry are categorized as

- chained read
- serial read
- directed read
- keyed read

All of these methods are available through the IMAGE library subroutine DBGET. The chained read method also requires the use of the DBFND subroutine to locate the record. Figure 4-1 illustrates the read methods using two data sets from the QA database.

Chained Read

The chained read method is used to retrieve the next or previous entry in the current chain. Chained reads occur on a detail data set only and require that the user call the DBFND routine prior to the first call to DBGET. The calling program specifies the name of the detail key item that defines the path to which the chain belongs and a value for the item. IMAGE determines which master set forms a path with the specified key item and locates the entry in that master set whose key item value matches the specified value. The entry located in the master data set contains pointers to the first and last entries in the desired chain and a count of the number of entries in the chain. This information is maintained by IMAGE and defines the current path.

If a program uses a chained read to read the TIME data set entries pertaining to project number 940071 shown in Figure 4-1, it must first call DBFND to locate the chain head in the PROJ data set. The program specifies the TIME data set, the PROJ# key item in the TIME data set and the value 940071 for that item. IMAGE uses a keyed read to locate the PROJ entry with a key item value of 940071. If the program then requests a forward chained read using DBGET, the entry in record 2, which is the beginning of the chain, is read. If a backward chained read is requested, the entry in record 6 is read.

Once a current path, and chain, is established for a data set, the calling program can use the chained read method of retrieving data. Both forward and backward chain reads are available. In either case, when the end of the chain is reached, DBGET returns a status code indicating this fact. To access entries from a different chain, the user must first call DBFND with the appropriate key item value or restore the current chain position through DBINF if that information was saved by a previous DBINF call prior to calling DBGET with a chain read.

Chain reads are particularly useful when the user wants to retrieve information about related events such as all failure records for assembly number 9206018001 in the QA database.

Re-Reading The Current Record

The DBGET subroutine allows the user to read the most recently accessed record again. A possible application is in a program that has unlocked the database and locked it again and needs to check if the contents of the current entry have been changed. Note that if a DBFND subroutine call, or a previous DBGET with a relative record number of zero, has been made, the current record is zero and a request to re-read the entry will cause an error.

Serial Read

In this mode of retrieval, IMAGE starts at the most recently accessed storage location for the data set, called the current record, and sequentially examines adjacent records until the next entry is located. Data items from this entry are returned to the calling program, and its location becomes the current record.

Both forward and backward serial reads may be used. Forward serial reads consist of retrieving the next greater-numbered entry and backward serial reads consist of retrieving the previous lower-numbered entry. If no entry is located, IMAGE returns an error code.

Since there is not a current record the first time a program requests an entry from a data set, a request for forward serial read causes IMAGE to search from record 1. Similarly, a backward serial retrieval begins at the highest numbered record. The user can reset the relative record number to zero by issuing a directed read with a record number of zero.

The entries connected by a broken line in Figure 4-1 are read by a program using the serial read method. If a forward serial read is performed on the TIME data set before any type of read, the entry in record number 2 is read. If another forward serial read is performed on the same data set, the entry in record 3 is read. On the other hand, if a serial read is performed and the current record is 7, the entry in record 10 is read. The next forward serial read on this data set returns an end-of-file condition.

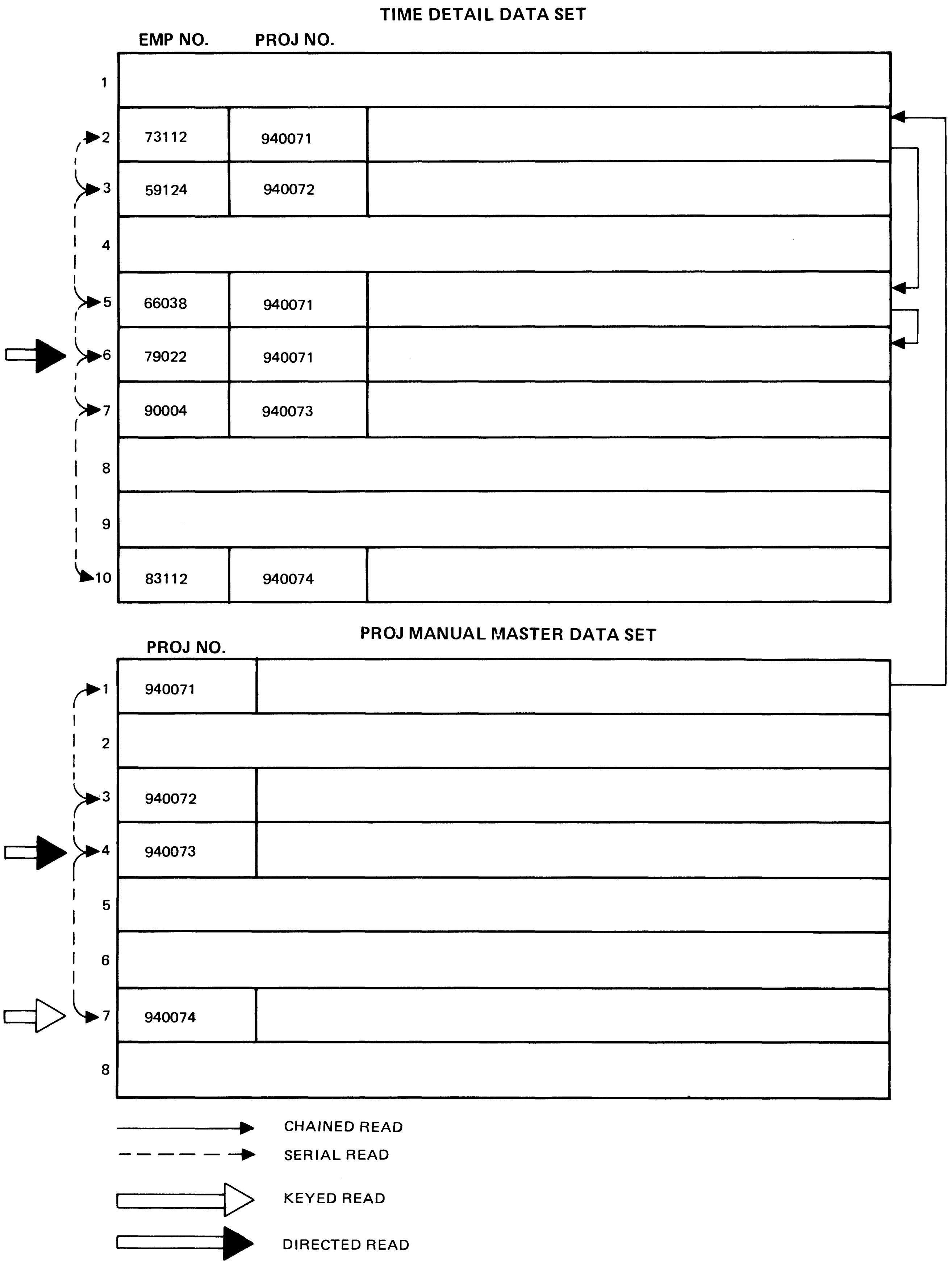


Figure 4-1. Reading Access Methods (DBGET).

The serial read method can be used with any type of data set and is very useful if most or all of the data in the data set is to be retrieved, for example, to be used in a report. The availability of serial reads effectively allows the use of a data set in the same way an FMP file would be used. Thus, the advantages of IMAGE database organization and the ease of serial access are available.

Directed Read

Another method of selecting the data entry to be read is to specify its record number. This method is called directed read. If an entry exists at the record address specified by the calling program, IMAGE returns the values for the data items requested in the calling program's buffer. If no such entry exists, the program is notified by an error code. If the user specifies a relative record address of zero, IMAGE resets the current record address to zero without performing a read. This is a convenient method of rewinding the data set.

This access method can be used with any type of data set and is useful in situations where the calling program has already determined the record number of the entry to be read. For example, if a program surveys several entries using another access method to determine which one it wants to use in a report, it can save each record number and use the record number of the entry it selects to read the entry again using the directed read method.

If a program performs a directed read of record 6 of the TIME data set, the entry marked with a solid black arrow in Figure 4-1 is read. If a directed read of the PROJ data set record 4 is performed, the entry in that set marked with the same type of arrow is read.

Keyed Read

The keyed read method allows the retrieval of an entry from a master data set by specifying a particular key item value. For example, the PROJ data entry for the project numbered 940074 shown in Figure 4-1, can be retrieved with this method since PROJ# is a key item in the PROJ data set. IMAGE locates the entry in the data set whose key item value matches the requested value. The exact technique used to perform keyed read is described in Chapter 7.

Keyed read can be used only with master data sets. It is very useful for retrieving a single entry for some special purpose. For example, a program used to get information about a particular assembly or project number could use keyed access to quickly locate the information in the QA database.

Updating Data

IMAGE allows the modification of the values of data items that are not key items if the mode with which the database was opened grants write access. Before calling the DBUPD subroutine to change the item values, the entry to be updated must be located. This is done via a DBGET call or a call to DBINF, if relative record information had previously been acquired and stored. This sets the current record address for the data set. The DBUPD subroutine uses the current record address to locate the data items whose values are to be changed.

When the program calls DBUPD, it specifies the data set name, a list of data items to be changed, and the name of a buffer containing values for the items. Before an entry is updated, DBUPD re-reads it so that the most current copy of the entry is updated. For example, if a program changes the number of failures of an assembly in the ASMBY data set of the QA database, the program can first locate the entry to be changed by calling DBGET in keyed read mode with the assembly number and then calling the DBUPD subroutine to change the value of the FAIL# data item in that entry.

Access Level And Open Mode

To update data items, the database must be opened in mode 1 or 3. In addition, if the open mode is 1, the database must have previously been locked. Caution must be taken when two or more programs have the database open in mode 1. If two programs attempt to update a data entry simultaneously, the first update may be lost. If this is likely to occur, the user should use the database locking routine DBLCK before using both the DBGET and DBUPD calls.

Any item to which the user has at least read access may be specified in the DBUPD item list. This convenience is allowed so that the same item list may be used for DBGET and DBUPD calls. However, the user may not attempt to change the value of an item for which he has read access only or is a key or sort item and any attempt to do so will result in an error.

Entering Data In The Database

Data is added to the database, one entry at a time, using the DBPUT subroutine. Data entries may be added to manual master and detail data sets. Entries are automatically added to automatic master data sets when entries are added to the associated detail data sets.

To add an entry, specify the data set name, a list of data items in the set, and the name of a buffer containing values for these items. Values must be supplied for key and sort items but are optional for other data items in the entry. If no value is supplied, the data item value is set to binary zeroes.

Sequence For Adding Entries

Before an entry can be added to a detail data set linked to a manual master data set, the manual master must contain an entry with a key item value equal to the one to be put in the detail. If more than one manual master is linked to the detail, each master must contain a key item value identical to the detail key item value for the corresponding path. To illustrate, consider the QA database again. Figure 4-2 contains sample data entries in four of the QA data sets.

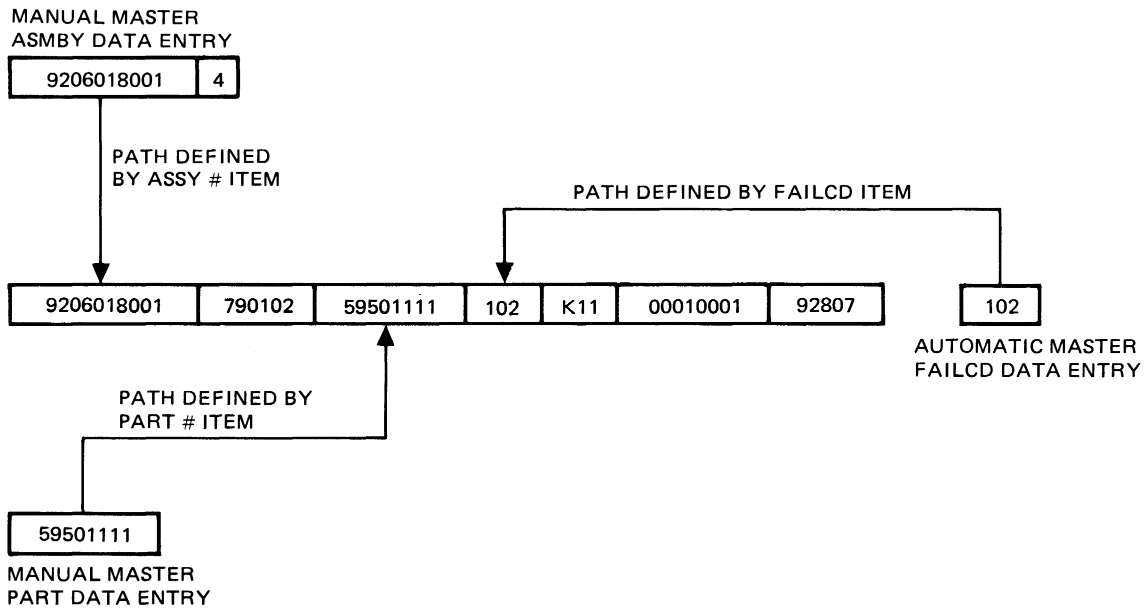


Figure 4-2. Sample Data Entries from QA Database

Before the FAIL data entry can be added to the data set, the ASMBY manual master data set must contain an entry with ASSY# equal to 9206018001 since ASSY# is the key item used to link to the FAIL detail. Similarly, the FAIL data set is linked to the PART manual master through the PART# key item so the entry with PART# equal to 59501111 must be added to PART before a test failure transaction for that PART# can be entered in FAIL.

Once the entry for assembly number 9206018001 has been entered, the next test failure for that assembly number can be entered in the FAIL detail set without changing the ASMBY master. This entry will be chained to the previous entry for the assembly. If part number 5950111 fails on a different assembly, the PART data set will not require any additional entries, but if the assembly's number is not yet in the ASMBY data set it must be added before entering the test failure transaction in FAIL.

Access Level And Open Mode

An entry cannot be added to a data set unless the access level established when the database is opened is as high as the highest write level in the data set. The database must also be opened with an open mode allowing entries to be added. These open modes are 1 and 3. If it is opened with access mode 1, DBLCK must be used to lock the data base before an entry can be added and DBUNL to unlock the database after one or all desired entries have been added.

Key Items

IMAGE performs checks on the values of key items before adding an entry to a data set. If the data set is a manual master, IMAGE verifies that the key item value is unique for the set, i.e., that no entry currently contains a key item with the same value. If the data set is a detail, IMAGE verifies that the value of each key item forming a path with a manual master has a matching value in that master. It also checks that there is room to add an entry to any automatic master data sets linked to the detail if a matching key item value does not exist.

Deleting Data Entries

To delete an entry from a data set, the entry to be deleted must first be located. Three methods are available: 1) read the entry with the DBGET subroutine directly, 2) read the entry with a combination of DBFND and DBGET, if it is advantageous to use chained read to find the entry or 3) restore chain information previously obtained with a call to the DBINF subroutine. Then call the DBDEL subroutine specifying the data set name. IMAGE verifies that the open mode and access level allow the deletion of the current entry of the specified data set.

If the detail data entry deleted is the only member of a detail chain linked to an automatic master, and all other chains linked to the same automatic master entry are empty, IMAGE automatically deletes the master entry.

If the data entry is a manual master set, IMAGE verifies that the detail chains associated with the entry's key item, if any, are empty. If not, it returns a non-zero status code to the calling program. For example, if a program attempts to delete the PROJ entry in Figure 4-1 that contains a PROJ# value of 940071, an error code is returned since a three entry chain still exists in the TIME detail data set.

To delete the PART data set entry with PART# equal to 59501234, the program can call DBGET in keyed read mode specifying the PART data set and the key item value 59501234. If the procedure executes successfully, the program then can call DBDEL specifying the PART data set to delete the current entry provided no chains in the related FAIL detail data set contain key item values of 59501234.

Access Level And Open Mode

The database must be opened with open mode 1 or 3. If it is opened with open mode 1, the DBLCK subroutine must be used to lock the data base before an entry can be deleted and DBUNL to unlock the database after one or all desired entries have been deleted.

An entry cannot be deleted from a data set unless the user access level established when the database was opened is as high as the highest write level in the data set.

Locking And Unlocking The Database

If a database is opened in mode 1, the DBLCK library subroutine can be used to gain temporary exclusive control of the database and the DBUNL routine to relinquish control. The database may be locked conditionally or unconditionally. If unconditional locking is requested, IMAGE returns control to the calling program only after exclusive control to the root file has been acquired. This occurs only after the user currently controlling the database, and any other users queued up waiting their turn, release control by unlocking the database.

If conditional locking is requested, IMAGE returns to the calling program immediately whether control has been acquired or not. In this case, the status code must be examined to determine whether or not the calling program has control.

When locking the database, the following special considerations apply:

- Failure of one process to unlock (or close) the database will cause all other processes attempting to lock the database to remain suspended until the process that has the database locked terminates.
- Unlocking results in relinquishing control of the database structure. That is, other processes may perform extensive deletions and additions on the database before the given process regains control. Therefore, although positional information relative to data sets, such as current record, current path, and so forth, is retained for the process during the unlock-lock sequence, the program must be prepared for the possibility that the state of the actual records has changed. For example, the current entry or next entry on the chain may have been deleted or replaced.

- The user may lock the database if it is open in mode 3. However, since mode 3 already implies exclusive access, the request will have no effect.

Closing The Database

After having completed all the operations on the database desired, use the DBCLS library subroutine to terminate access to it. When DBCLS is used for this purpose, all data set files and the root file are closed. The DBCLS routine can also be used to close a particular data set.

It is important to close the database before terminating a program to ensure that the files are closed in an orderly manner. Once a database has been closed, it may be accessed again by that program only by reissuing a request to open it.

Although the facility is available to close one particular data set, in general, there should be no need to do so.

Checking The Status Of An Operation

Each time a subroutine is called, IMAGE returns status information in a buffer specified by the calling program. The first word of this buffer is always a status code, which is zero if the operation was successful and non-zero if the operation was not completed. This status word should be checked after every operation. An error which is not caught at the time it occurred could cause additional problems as the program progresses.

In addition to the error status word, other information may be passed to the user in the status buffer. This allows IMAGE to provide the user with current information about the database as each operation continues.

Under normal conditions, those words in the status array which do not have information placed in them as a result of the current operation will remain unchanged from previous operations. However, remote data base access requires a ten word status array and any undefined words in that array may be overwritten as a result of remote operations.

Call Parameters

In the calls to IMAGE subroutines, many parameters are used in more than one subroutine with similar or identical meanings. The following paragraphs describe these parameters and their general use. Specific values they may take for specific calls are listed under the descriptions for the calls.

Because of the differences in language requirements, the same information might be passed differently from a FORTRAN, PASCAL, or Assembly Language progr than from a BASIC program. For this reason, the majority of the parameter descriptions address FORTRAN, PASCAL, and Assembly Language usage separate from BASIC usage.

In general, the first four parameters of each library call are the database name, a data set name or number, a mode parameter and the name of a buffer for the return of status information. The mode parameter is an integer variable which has different possible values for different calls. Exceptions to the general rule are in DBOPN, whose second parameter is the level word and DBINF whose second parameter could be either a data set or item name or number. Some calls may have additional parameters beyond these four.

In FORTRAN and Assembly Language usage, names are passed as integer arrays, numbers as one word integers. Values passed in an integer array are left justified, padded with blanks, if necessary. From BASIC programs, names are passed in character string arrays. The following is a general description of the parameters used in the IMAGE subroutine library calls.

IBASE

In FORTRAN, PASCAL, and Assembly Language usage, IBASE is an integer array which contains the data base FMP file namr. The first word is either a DS node number (if accessing a database at a remote site), or two ASCII blanks. Words two through the end of the array contain at least the database and the security code, separated by a colon and terminated by a blank or semi-colon. An example of IBASE is this:

```
     DIMENSION IBASE(6)
     DATA IBASE /101,2HQA,2H:1,2H00,2H:4,2H3;/
```

This example would allow access of a database of name QA, a security code of 100 and a cartridge reference number of 43 at a DS remote node of 101. Note that a node number of 8224, although legal to DS would be interpreted as two blanks and therefore cannot be used for remote database access.

From a BASIC program, IBASE would be a one-dimensional string array containing two ASCII blanks followed by the database namr. Remote database access is not supported from BASIC programs, so the first two bytes cannot contain a DS node number, but must be blank.

Once a successful open has been performed on the database specified by IBASE in the DBOPN call, IMAGE replaces the contents of IBASE with a database number. This will be used in subsequent library calls. For this reason, the contents of IBASE should not be modified as long as the database is open. Once the database is closed, the IBASE array is returned to its original values before the DBOPN call, and thus can be used in a subsequent open.

ILEVL

ILEVL is used to pass the level code word, which will determine the user's level of access. This level word, supplied only when opening the database, will detemine what items in the database a user can access. ILEVL is, for FORTRAN and Assembly Language users, an integer array of three words, blank-filled if necessary. For BASIC users, it is a one-dimensional string array.

ID

The ID parameter is used to supply a data set name for database manipulation calls. For DBINF calls, ID may also be a data item name. For BASIC calls, the name is in a one-dimensional string array. For FORTRAN and Assembly Language usage, names are passed in an integer array of three words, blank-filled if necessary. In FORTRAN and Assembly Language programs, ID may also contain an integer which is a data set or data item number.

IMODE

IMODE is used to pass information to the subroutine via an integer variable. It may be an access mode parameter, an information series indicator or a read option. For some calls, IMODE must contain the value 1.

ISTAT

ISTAT is a array that IMAGE uses to return status information to the user. In FORTRAN and Assembly Language calls, ISTAT is ten words long. In BASIC calls, the length is variable, ranging from one to six elements. In all cases, if the operation requested was unsuccessful, a non-zero code is returned in the first word or element of the array. If the operation is successful, the first word or element of the ISTAT array is set to zero. Depending on the operation, additional information may be passed via the remainder of the array. Specific information on returned information is detailed under each specific call description.

IBUF

IBUF is a buffer used to pass information to and from the user. The size of the buffer is variable based on the information to be passed. Use of the buffer is dependent on the operation being performed and is described in more detail under the pertinent call description.

ITEM

ITEM is an integer array of at least three words in FORTRAN or Assembly Language or a one-dimensional string array in BASIC containing the name of the data item on which a search is to be made. For FORTRAN and Assembly Language, ITEM may alternately be an integer variable containing a data item number.

IARG

IARG is a real or integer variable or an integer array or a one-dimensional string array containing a value given to a key item or a doubleword integer variable containing a relative record number.

LIST

LIST is an integer array used in FORTRAN and Assembly Language calls, which contains a list of data item names or numbers. A list of data item names is a set of data item names separated by commas and terminated by a blank or semi-colon. A list of data item numbers is an integer N followed by a set of N data item numbers, where N is from 1 to the number of items in the data set being accessed.

Two special constructs are allowed. If the first word of the list array contains "@ " ('at' sign followed by a blank) the entire data record part of the data entry is read into the user buffer (excluding the media record part). If the first word of the list array contains either an ASCII "0 " (zero followed by a blank) or the integer value zero, no data is returned to the user. For a data item name or number list, the order in which items are specified is the order in which the values will be stored or returned in the IBUF array, regardless of the order in which they are stored in the data entry.

An example of LIST is the following:

```
     DIMENSION LIST(9)
     DATA LIST/2HAS,2HSY,2H#,,2HPA,2HRT,2H#,,2HEM,2HP#,2H; /
```

This list contains three data item names - ASSY#, PART# and EMP#.

NAME-LIST

NAME-LIST is a one-dimensional string array used only in BASIC calls containing a list of one or more data item names, separated by commas and terminated by a blank or semi-colon. No more than ten names may appear in a name-list. In a DBGET operation, the list may not contain more than nine names. NAME-LIST is used in retrieval, add and update operations to specify the data items to be affected as a result of the operation. Special constructs such as explained in the paragraph on LIST, cannot be used in NAME-LIST.

```
    DIM G$[41]
    DATA "ASSY#,DATE,PART#,FAILCD,REFDES,PNTS,EMP#;"
    READ G$
```

VALUE-LIST

VALUE-LIST is a list of one or more variables separated by commas and corresponding in type and number to the data items given in NAME-LIST. The variables in VALUE-LIST are the values either retrieved from or added to the data-base as values for those items named in the NAME-LIST.

In the following example of a BASIC DBGET call, I$ is the NAME-LIST containing two item names, and Y$ and F1 are the VALUE-LIST.

```
     DIM I$[12]
     DATA "ASSY#,FAIL#"
     READ I$
              .
              .
              .
     CALL DBGET(Q$,S$,M7,S[1],A$,I$,Y$,F1)
```

Using The IMAGE Library Procedures

In the following discussion, the calling parameters for each IMAGE subroutine are defined in the order in which they appear in the call statement. Each parameter must be included when a call is made since their meaning is determined by their position.

Table 4-1 illustrates the forms of the call statements for the three languages that can be used to call the procedures. Chapter 5 contains examples of using the IMAGE routines with these languages.

Table 4-1. Calling an IMAGE Subroutine

```
FORTRAN    CALL name(parameter,parameter,...,parameter)

PASCAL     name (parameter,parameter,...,parameter)

Assembly   JSB   name
           DEF   *+(n+1)
           DEF   parm1
           DEF   parm2
               .
               .
               .
           DEF   parmn

BASIC      lineno CALL name(parameter,parameter,...,parameter)
```

Unused Parameters

When calling some procedures for a specific purpose, one of the parameters may be ignored. However, it must be listed in the call statement. An application program may find it useful to set up a variable named DUMMY to be listed as the unused parameter in these situations, as a reminder that the value of the parameter does not affect the subroutine call.

IMAGE Subroutine Calls

The following sections detail each IMAGE subroutine, giving the parameters used, their possible values, and information about the use of the subroutine. The sections are ordered alphabetically for easy reference. Should additional information be needed, each call description details where such information can be found.

DBCLS

DBCLS(ibase,id,imode,istat)

Terminates access to a database or closes a data set.

PARAMETERS:

ibase	is the array used as the IBASE parameter when opening the database. The array must not have been altered since issuing the call to DBOPN.
id	is an array containing the name of the data set to be closed or a variable containing the data set number, if mode equals 2. If mode equals 1, this parameter is ignored.
imode	is an integer variable indicating the type of close desired. imode = 1: the database is closed. imode = 2: the data set referenced by the ID parameter is closed.
istat	is an array in which IMAGE returns status information about the operation. If the operation was successful, the first word or element of the array is zero.

CLOSING A DATABASE

When a successful close is performed on a database, the database number which was written into the IBASE array is removed and the array is returned to its original value before the database open. This allows the IBASE array to be reused for a subsequent open operation. In addition, the root file and all data sets are closed.

CLOSING DATA SETS

Although a program should close the database when it is finished using the data, closing the data set is an optional function. Closing a data set allows space for DCBs for additional data sets to be opened without having to save the previously used DCB. If a program accesses one data set for a period of time and then does not access it but begins to access another, closing the data set is recommended.

STATUS CODES:

The following are status codes which are returned from a DBCLS operation. For more information about any of these codes, see Appendix B.

CODE	MEANING
100	Illegal data set reference (mode 2 close only)
103	Illegal database parameter or database not open
115	Illegal DBCLS mode
135	Data base is locked to the user and attempts to unlock it are unsuccessful
137	Illegal resource number usage. DBCOP was unable to release the database's resource number when this was the last DBCLS on a database opened in mode 1
140	DBCLS was unable to schedule DBCOP to remove its entry from the coordinating table
160	IMAGE data structures are corrupt
161	Bad function code passed to DBCOP

TEXT DISCUSSION:

Page 4-15

PARAMETER DISCUSSION:

IBASE	Page 4-17
ID	Page 4-18
IMODE	Page 4-18
ISTAT	Page 4-18

EXAMPLES:

FORTRAN	Page 5-15
PASCAL	Page 5-35
Assembler	Page 5-40
BASIC	Page 5-52

DBDEL

DBDEL(ibase,id,imode,istat)

Deletes the current entry from a data set. The database must be open in mode 1 or 3 and the level of access granted at open time must allow deleting from the data set.

PARAMETERS:

ibase	is the array used as the IBASE parameter when opening the database. The array must not have been altered since issuing the call to DBOPN.
id	is an array containing the name of the data set from which the entry is to be deleted. In FORTRAN or assembly language, it may also be a data set number.
imode	must be an integer of value 1.
istat	is an array in which IMAGE returns status information. If the procedure executes successfully, the first word or element of the array is zero. The remainder of the array is meaningless.

MASTER DATA SETS

When deleting entries from manual master data sets, the following rule applies. All pointer information for chains associated with the entry must indicate that the chains are empty. In other words, there must not be any detail entries chained to the master entry.

DETAIL DATA SETS

IMAGE performs the required changes to chains, including the chain heads in related master data sets necessary to indicate that the entry is deleted. If the last member of a chain linked to an automatic master entry has been deleted, DBDEL also deletes the automatic master entry containing the chain head.

STATUS CODES:

The following are status codes which are returned from a DBDEL operation. For further information about these codes, see Appendix B.

CODE	MEANING
100	Illegal data set reference
103	Illegal database parameter or database not open
104	Data base not enabled for deleting (opened in mode 8)
108	Request directed at an automatic master
113	Master entry still contains non-zero path counts manual master only)
114	Current record is empty
115	Invalid DBDEL mode (must be 1)
118	Data set not write enabled by user's access level
154	Data base corrupt - bad chain pointers
157	No current record in data set
159	Data base opened in mode 1 but not previously locked to user
160	IMAGE data structure corrupt

TEXT DISCUSSION:

Page 4-13

PARAMETER DISCUSSION:

IBASE	Page 4-17
ID	Page 4-18
IMODE	Page 4-18
ISTAT	Page 4-18

EXAMPLES:

FORTRAN	Page 5-12
PASCAL	Page 5-35
Assembler	Page 5-39
BASIC	Page 5-49

DBFND

DBFND(ibase,id,imode,istat,item,iarg)

Locates the master set entry that matches the specified key item value and sets up pointers to the detail data set chain in preparation for chained access to the data entries which are members of the chain.

PARAMETERS:

ibase	is the array used as the IBASE parameter when opening the database. The array must not have been altered since issuing the call to DBOPN.
id	is an array containing the name of the detail data set to be accessed. If using FORTRAN or Assembly Language, ID may also be a data set number.
imode	must be an integer of value 1.
istat	is an array in which IMAGE returns status information about the procedure. If the procedure executes successfully, the status array from FORTRAN, PASCAL, or Assembly Language has the following contents:

Word	Contents
1	Zero
2	Zero
3-4	Doubleword integer current record number set to zero
5-6	Doubleword integer count of number of entries in chain
7-8	Doubleword integer record number of last entry in chain
9-10	Doubleword integer record number of first entry in chain

For BASIC calls, the status array has the following contents:

Element	Contents
1	Zero
2	Zero
3	Current record number set to zero
4	Count of the number of entries in the chain
5	Record number of last entry in chain
6	Record number of first entry in chain

item	is an array containing the name of the detail data set's key item that defines the chain desired. If calling DBFND from FORTRAN or Assembly Language, it may also be a data item number.
iarg	contains a key item value to be used to locate the desired chain head in the master data set.

DATA SET CHAINS

The current values of chain count, backward pointer, and forward pointer for the detail data set referenced in ID are replaced by the corresponding value from the chain head. A current path number, which is maintained internally, is set to the new path number and the current record number for the data set is set to zero. Refer to Chapter 7 for further information about chain heads and internally maintained data set information.

Note that although a master set entry exists with the specified key item value, the data set chain may be empty. In this case, a DBFND operation will return a status code of 156.

STATUS CODES:

The following are status codes which are returned from DBFND. For more information about these codes, see Appendix B.

CODE	MEANING
100	Illegal data set reference
101	Illegal data item reference
102	Data item specified is not a key item
103	Illegal database parameter or database not open
107	No master record with key value
115	Illegal DBFND mode
120	Data set specified is not a detail data set
121	Detail data set has no paths
154	Data base corrupt, chain pointers invalid
156	No detail records on chain
160	IMAGE data structures are corrupt
162	Missing parameter in DBFND call

TEXT DISCUSSION:

Page 4-5

PARAMETER DISCUSSION:

IBASE	Page 4-17
ID	Page 4-18
IMODE	Page 4-18
ISTAT	Page 4-18
ITEM	Page 4-19
IARG	Page 4-19

EXAMPLES:

FORTRAN	Page 5-4
PASCAL	Page 5-35
Assembler	Page 5-38
BASIC	Page 5-43

DBGET (FORTRAN And Asembly Language)

DBGET(ibase,id,imode,istat,list,ibuf,iarg)

Provides seven different methods for reading the data items of a specified entry. The DBGET call from FORTRAN, PASCAL, and Assembly Language differs from that of BASIC in the order of the parameters.

PARAMETERS:

ibase	is the array used as the IBASE parameter when opening the database. The array must not have been altered since issuing the call to DBOPN.
id	is an array containing the name of the detail or master data set to be read or an integer variable containing the data set number. Unless otherwise stated, either a detail or master data set is acceptable.
imode	is an integer between 1 and 7, inclusive, which indicates the reading method. The methods are:

Mode	Method
1	Re-read Read the entry at the current record location. The IARG parameter is ignored in this mode.
2	Serial Read Search the data set from the current record plus one in the direction of increasing record numbers until the first non-empty record is found and read. The IARG parameter is ignored in this mode.
3	Backward Serial Read Search the data set from the current record minus one in the direction of decreasing record numbers until the first non-empty record is found and read. The IARG parameter is ignored in this mode.
4	Directed Read Read the entry, if it exists, at the record address specified in the IARG paramter. IARG is treated as a doubleword integer record number.
5	Chained Read Read the next entry in the current chain, the entry referenced by the forward chain pointer. In this mode, ID must be the name or number of a

detail data set. The IARG parameter is ignored for this mode.

6 Backward Chained Read
Read the previous entry in the current chain, the entry referenced by the backward chain pointer. In this mode, ID must be a detail data set name or number. The IARG parameter is ignored for this mode.

7 Keyed Read
Read the entry with a key item value that matches the value specified in IARG. In this mode, ID must be a manual master data set.

istat is a ten-word array in which IMAGE returns status information about the operation. If the procedure executes successfully, the status array has the following contents:

Word	Contents
1	zero
2	word length of data transferred
3-4	doubleword integer record number of entry read
5-6	doubleword integer zero
7-8	doubleword integer record number of the predecessor of the current record along the current chain (chained read only)
9-10	doubleword integer record number of the successor of the current record along the current chain (chained read only)

list is an array containing a data item name list or a data item number list.

ibuf is an array in which DBGET returns the concatenated values of the items specified in LIST. The order of the values in IBUF is the same as the order of the item names or numbers specified in LIST, regardless of the order of the items in the data entry.

iarg is a doubleword integer containing a record number for a directed read or a variable containing a key item value for a keyed read.

DATA SET POINTERS

After doing a chained read, the backward and forward pointers for the data set are replaced by the current record's chain pointers from the entry just read. Before the first chained read, words 7 through 10 of the status array still reflect their value from the DBFND operation. After the first chained read, the current record number appears in words 3 and 4. The preceding record number (word 7 and 8) is zero and the succeeding record number (word 9 and 10) is the next record in the chain. When the last record in the chain is read, zero is the succeeding record number.

For all other types of reads, word 5 through 10 of the status array will be undefined.

If a directed read is made with an IARG value of zero, the current record pointer is reset to the beginning of the file.

STATUS CODES:

The following are status codes which are returned from DBGET. For more information about these codes, see Appendix B.

CODE	MEANING
12	Beginning of file or end of file encountered
100	Illegal data set reference
101	Illegal item list
103	Illegal database parameter or database not open
107	No master entry with key value (keyed read mode only)
111	Path has not been initialized for a chained read, or record number on a directed read is illegal
114	Record accessed is empty
115	Invalid DBGET mode
118	Key item inaccessible
120	Data set specified is not a detail (chained read only)
123	Data set specified is not a master (keyed read only)
154	Data base corrupt, bad chain pointers
155	Beginning or end of chain
157	No current record (serial read or re-read only)
160	IMAGE data structures corrupt
162	Missing parameter in DBGET call

TEXT DISCUSSION:

Page 4-5

PARAMETER DISCUSSION:

IBASE	Page 4-17
ID	Page 4-18
IMODE	Page 4-18
ISTAT	Page 4-18
LIST	Page 4-19
IBUF	Page 4-18
IARG	Page 4-19

EXAMPLES

FORTRAN	Page 5-6
PASCAL	Page 5-35
Assembler	Page 5-39

DBGET (BASIC)

DBGET(ibase,id,imode,istat,iarg,name-list,value-list)

The difference between the DBGET call for BASIC and FORTRAN, PASCAL, or Assembly Language is in the order of the parameters. The following discussion gives the proper order of the parameters for a BASIC DBGET call, but the user is referred back to the previous discussion of DBGET for more information.

PARAMETERS:

ibase	is the array used as the IBASE parameter when opening the database. This array must not have been altered since issuing the call to DBOPN.
id	is a character string containing a detail or master data set name.
imode	is an numeric variable containing the mode of the get. For possible values and their meanings, see the previous DBGET section.
istat	is an array in which status information is returned to the calling program. For a successful completion of the operation, the following information is returned in the ISTAT array.

Element	Contents
1	Zero
2	Word length of data transferred
3	Record number of current record
4	Zero
5	Record number of the predecessor of the current record along the current chain
6	Record number of the successor of the current record along current chain.

iarg	is an numeric variable containing a record number for a directed read or a variable or array containing a key item value for a keyed read.
name-list	is a character string containing a list of one or more data item names separated by commas and terminated by a blank or semi-colon. No more than nine names may appear in a name-list for a DBGET. Special constructs are not allowed in NAME-LIST.

value-list is a list of one or more string or numeric variables corresponding in type and number to the data items given in the NAME-LIST. DBGET reads the data items specified in name and stores the values of these data items into the variables appearing in the VALUE-LIST.

STATUS CODES:

The following are status codes which are returned from a DBGET operation. For more information about these codes, see Appendix B.

CODE	MEANING
12	Beginning of file or end of file encountered
100	Illegal data set reference
101	Illegal item list
103	Illegal database parameter or database not open
107	No master entry with key value (keyed read mode only)
111	Path has not been initialized for a chained read, or record number on a directed read is illegal
114	Record accessed is empty
115	Invalid DBGET mode
118	Key item inaccessible
120	Data set specified is not a detail (chained read only)
123	Data set specified is not a master (keyed read only)
154	Data base corrupt, bad chain pointers
155	Beginning or end of chain
157	No current record (serial read or re-read only)
160	IMAGE data structures corrupt
162	Missing parameter in DBGET call

In addition, BASIC may return the following status codes:

CODE	MEANING
302	Too many names in NAME-LIST (more than nine)
303	Invalid name in NAME-LIST
304	The type or length of a variable in the VALUE-LIST does not match the type or length of its corresponding item in the NAME-LIST
306	Invalid relative record number (non-numeric parameter)

TEXT DISCUSSION:

Page 4-5

PARAMETER DISCUSSION:

IBASE	Page 4-17
ID	Page 4-18
IMODE	Page 4-18
ISTAT	Page 4-18
IARG	Page 4-19
NAME-LIST	Page 4-20
VALUE-LIST	Page 4-20

EXAMPLES:

BASIC	Page 5-45

DBINF

DBINF(ibase,id,imode,istat,ibuf)

Provides information about the database being accessed. The information returned is restricted by the access level established when the data base is opened. Data items, data sets, or paths of the database which are inaccessible with the user's access level are considered to be non-existent. A data item is inaccessible if the read level of the item is higher than the level used to open the database. A data set is inaccessible if all items in the set are inaccessible.

PARAMETERS:

ibase	is the array used as the IBASE parameter when opening the database. The array must not have been altered since issuing the call to DBOPN.
id	is an array containing a data set or data item name or, if FORTRAN or Assembly Language, a data set or data item number.
imode	is an integer indicating which type of information is desired. The available modes and information supplied with each are described in the tables 4-2 and 4-3. The general form of the mode parameter is data item modes are 1nn, data set modes are 2nn, path modes are 3nn and chain modes are 4nn.
istat	is an array in which IMAGE returns status information about the operation. After successful completion of the operation, the following information is returned.

Element	Contents
1	Zero
2	Length of the information in IBUF

ibuf	is an array in which the requested information is returned. The contents of the IBUF array vary according to the IMODE parameter used. They are also described in tables 4-2 and 4-3.

Table 4-2. MODE and ID Values and Results for FORTRAN and Assembly Language

mode	PURPOSE	*id*	*ibuf* ARRAY CONTENTS	COMMENTS
101	Defines type of access available for specific item.	data item name or number	word 1 ± data item number	If negative, user has read and write access. If positive, user has read access only.
102	Describes specific data item.	data item name or number	word 1 . . 8 data item name 9 data type Δ 10 element length 11 element count 12 0 13 0	Left-justified and padded with blanks, if necessary. (I, ,R, X,) Δ indicates blank If data type is x, length is in bytes, else length is in words.
103	Identifies all data items available in data base and type of access allowed.	(ignored)	word 1 n 2 ± data item number . . . . n+1 ± data item number	n = number of data items available Arranged in data item number order. If positive, read-only access. If negative, read and write access.
104	Identifies all data items available in specific data set and type of access allowed.	data set name or number	(Same as *mode* 103)	(Same as *mode* 103 *except* arranged in order of occurrence in data entry.)
201	Defines type of access available for specific data set.	data set name or number	word 1 ± data set number	If negative, entries can be added or deleted.
202	Describes specific data set	data set name or number	word 1 . 8 data set name 9 set type Δ 10 entry word-length 11 0 12 0 13 0 14 15 number of entries in set 16 17 capacity of set	Left-justified and padded with blanks, if necessary. (M,A,D) Δ indicates blank integers doubleword integers

Table 4-2. MODE and ID Values and Results for FORTRAN and Assembly Language (con't.)

mode	PURPOSE	*id*	*ibuf*	ARRAY CONTENTS	COMMENTS
203	Identifies all data sets available in data base and type of access allowed.	(ignored)	word 1 2 . . . n+1	 n ± data set number . . . ± data set number	n = number of data sets available Arranged in data set number order. If positive, entries may not be added or deleted. If negative, entries may be added or deleted.
204	Identifies all data sets available which contain specified data item and type of access allowed.	data item name or number	(Same as *mode* 203)		(Same as *mode* 203)
301	Identifies paths defined for specified data set.	data set name or number	word 1 2 3 4 . . . . . 3n−1 3n 3n+1	 n data set number search item number sort item or Ø . . . . . data set number search item number sort item or Ø	n = number of paths Repeat for each path. If *id* refers to master, set number is for detail. If *id* refers to detail, set number is for master. Item numbers identify items in detail. Path designators presented in order of their appearance in schema.
302	Identifies search item for specified data set.	master data set name or number	word 1 2	 search item number 0	zero if inaccessible

Table 4-2. MODE and ID Values and Results for FORTRAN and Assembly Language (con't.)

mode	PURPOSE	*id*	*ibuf* ARRAY CONTENTS	COMMENTS
401	Obtain current chain information	detail data Set name or number	word 1, 2: last record accessed record number 3, 4: previous record in chain record number 5, 6: next record in chain record number 7: current chain path no.	doubleword integers (words 1–6)
402	Restores previously saved chain information as current chain in run table	detail data set name or number	Same as *mode* 401	

Table 4-3. MODE and ID Values and Results for BASIC

mode	PURPOSE	*id*	*ibuf* ARRAY CONTENTS		COMMENTS
101	Defines type of access available for specific item.	data item name	character		If negative, user has read and write access. If positive, user has read access only.
			1	±	
			2	Δ	
102	Describes specific data item.	data item name	character		
			1	item type	I, R or X
			2	,	(comma)
			3	element length	in words for type I or R, else bytes
			4		
			5		
			6	,	(comma)
			7	element count	
			8		
			9		
103	Identifies first 36 items available in data base	(ignored)	character		
			1	number of items	Arranged in data item number order maximum of 36 names in 255 character buffer
			2		
			3		
			4	,	
			5	data item	
			6		
			7		
			8		
			9		
			10		
			11	,	(comma)
104	Identifies first 36 items available in specific data set and type of access allowed.	data set name	(Same as *mode* 103)		(Same as *mode* 103 *except* arranged in order of occurrence in data entry.)
201	Defines type of access available for specific data set.	data set name	character		If negative, entries can be added or deleted.
			1	±	
			2	Δ	
202	Describes specific data set	data set name	character		
			1	data set type	M, A or D
			2	,	(comma)
			3	length of entry	in words
			⋮		
			6		
			7	,	(comma)
			8	number of entries in data set	
			⋮		
			17		
			18	,	(comma)
			19	data set capacity	
			⋮		
			28		

Table 4-3. MODE and ID Values and Results for BASIC (con't.)

mode	PURPOSE	*id*	*ibuf* ARRAY CONTENTS	COMMENTS
203	Identifies first 36 sets available in data base.	(ignored)	character number of data sets , 5 . . 10 data set name 11 , . . .	maximum of 36 data set names in 255 characters
204	Identifies first 36 sets available which contain specified data item.	data item name	(Same as *mode* 203)	(Same as *mode* 203)
301	Identifies paths defined for specified data set.	data set name	character 1 2 number of paths 3 , 4 . . 9 related data set 10 , 11 . . 16 key item name 17 , 18 . . 23 sort item name (or blanks) 24 , 25 repeat from set name	 (comma) (comma) (comma) (comma)
302	Identifies search item for specified data set.	master data set name	character 1 . . 6 key item name	blank if unaccessible

Table 4-3. MODE and ID Values and Results for BASIC (con't.)

mode	PURPOSE	*id*	*ibuf* ARRAY CONTENTS	COMMENTS
401	Obtain current chain information.	detail data set name	same as FORTRAN or Assembly language Note: The data in this array is for internal IMAGE use only. Its format is unusable by the BASIC program.	
402	Restores previously saved chain information as current chain in Run Table.	detail data set name	(Same as *mode* 401)	

STATUS CODES:

The following are the status codes returned from DBINF. For more information about these codes, see Appendix B.

CODE	MEANING
103	Illegal database parameter or database not open
123	Data set is not a master data set (mode 302 only)
124	Illegal DBINF mode
125	Illegal data item or data set reference
160	IMAGE data structures are corrupt
162	Missing parameter in DBINF call

In addition, BASIC may return the following status code:

CODE	MEANING
324	Invalid DBINF mode

TEXT DISCUSSION:

Page 4-3

PARAMETER DISCUSSION:

IBASE	Page 4-17
ID	Page 4-18
IMODE	Page 4-18
ISTAT	Page 4-18
IBUF	Page 4-18

EXAMPLES:

FORTRAN	Page 5-3
PASCAL	Page 5-35
Assembler	Page 5-38
BASIC	Page 5-42

DBLCK

DBLCK(ibase,id,imode,istat)

Provides temporary exclusive control of a database by locking the root file. A redundant call while in open mode 1 or a call while in open mode 3 is ignored.

PARAMETERS:

ibase	is the array used for the IBASE parameter when the database was opened. This array must not have been altered since issuing the call to DBOPN.
id	is unused. Use a dummy variable or any ID variable previously used.
imode	is an integer defining the mode of locking. imode=1 is lock with wait. Control is returned to the calling program only after exclusive control of the database has been acquired. imode=2 is lock without wait. Control is returned to the calling program immediately with a code in the first word of the ISTAT array. If control has been acquired, the status word is zero. If control was not available, the status word contains a non-zero value.
istat	is an integer array containing the status information. If a lock has been successful, the first word or element of the array is zero.

STATUS CODES:

The following are status codes returned from DBLCK. For more information about these codes, see Appendix B.

CODE	MEANING
103	Illegal database parameter or database not open
115	Illegal lock mode
134	Data base not enabled for locking (open in mode 8)
136	Data base locked to another user (lock mode 2 only)
137	Illegal resource number usage. Resource number has been released or written over.

TEXT DISCUSSION:

Page 4-14

PARAMETER DISCUSSION:

IBASE	Page 4-17
IMODE	Page 4-18
ISTAT	Page 4-18

EXAMPLES:

FORTRAN	Page 5-14
PASCAL	Page 5-35
Assembler	Page 5-40
BASIC	Page 5-50

DBOPN

DBOPN(ibase,ilevl,imode,istat)

Initiates access to the database and establishes the access mode and level for all subsequent database access.

PARAMETERS:

ibase	is an array containing a string of ASCII characters which is the name of the database to be opened. If the database successfully opened, DBOPN writes the database number into the array. In all subsequent accesses, the IBASE array must contain this number. Therefore, it should not be altered while the database is open.
ilevl	is an array containing the level word which establishes the access level of this user.
imode	is an integer variable containing the open mode. The modes available are

1	Shared read/write with locking access. All other users must have opened the database in mode 1.
3	Exclusive read/write access
8	Shared read access. All other users must have opened the database in mode 8.

istat is an array into which DBOPN returns the status information. If the operation is successful, the status array contains the following information:

Word	Contents
1	zero
2	user's assigned access level
3	length of the Run Table in words

STATUS CODES:

The following are the status codes returned from DBOPN. For more information about these codes, see Appendix B.

CODE	MEANING
103	Illegal database parameter
115	Illegal open mode
116	File specified by IBASE parameter is not a root file

117	Bad root file security code
119	Root file cannot be found
128	Not enough space for IMAGE data buffers
129	Root file opened exclusively
131	No room in co-ordinating table; 20 databases open already
132	No resource number available for a database being opened in mode 1
140	Cannot schedule DBCOP
150	Database is already open to user
152	Database open to another user in an incompatible mode
153	User has no access to any item or set in one database with a given level
160	IMAGE data structures are corrupt
161	Internal subroutine passed DBCOP an illegal function code

In addition, BASIC may return the following status codes:

CODE	MEANING
310	Invalid IBASE parameter (first two characters not blanks)

TEXT DISCUSSION:

Page 4-1

PARAMETER DISCUSSION:

IBASE	Page 4-17
ILEVL	Page 4-17
IMODE	Page 4-18
ISTAT	Page 4-18

EXAMPLES:

FORTRAN	Page 5-2
PASCAL	Page 5-35
Assembler	Page 5-38
BASIC	Page 5-41

DBPUT

```
DBPUT(ibase,id,imode,istat,      list     ,    ibuf      )
                               name-list     value-list
```

Adds new entries to a manual master or detail data set. The database must be open in mode 1 or 3 and the level of access granted at open time must allow addition. If the database is open in mode 1, it must have been locked with a DBLCK call.

PARAMETERS:

ibase	is the array used as the IBASE parameter when opening the database. The array must not have been altered since issuing the call to DBOPN.
id	is an array containing the name of the detail or manual master data set to which the entry is to be added. If the call is from FORTRAN or Assembly Language, it may also be a data set number.
imode	must be an integer equal to 1.
istat	is an array in which IMAGE returns status information. If the operation is successfully completed from a FORTRAN or Assembly Language program the status array will contain the following information:

Word	Contents
1	Zero
2	Word length of the entry in IBUF array
3-4	Doubleword record number of the new entry
5-6	Doubleword count of entries in the current chain (if a detail data set) or synonym chain (if manual master data set)
7-8	Doubleword record number of predecessor in the current chain (if detail data set) or synonym chain (if manual master data set)
9-10	Doubleword record number of successor in current chain (if a detail data set) or zero (if a manual master data set)

For a successful call from a BASIC program, the status array will contain the following:

Element	Contents
1	Zero
2	Word length of data entry in the print-list
3	Record number of new entry
4	Count of entries in current chain (if a detail data set) or synonym chain (if a manual master data set)
5	Record number of predecessor in current chain (if a detail data set) or synonym chain (if a manual master data set)
6	Record number of successor in current chain (if a detail data set) or zero (if a manual master data set)

list OR name-list is an array containing a set of data item identifiers. The new entry contains values supplied in the IBUF or VALUE-LIST array. Fields of unreferenced items are filled with binary zeroes. NAME-LIST, used in BASIC calls, cannot contain special constructs.

ibuf OR value-list is an array or list containing the data item values to be added. For FORTRAN, PASCAL, or Assembly Language, IBUF is an array. The values are concatenated in the same order as their data item identifiers. For BASIC usage, VALUE-LIST is a list of values. The number of words or elements for each value must correspond to the number required by its type.

MASTER DATA SETS:

When adding entries to master data sets the following rules apply:

- o The data set must be a manual master
- o The key item must be referenced in the LIST array or NAME-LIST and its value in the IBUF array or VALUE-LIST must be unique in relation to other entries in the master.
- o There must be space in the master set to add a new entry.
- o The order of data item values in the new entry is determined by the set definition in the schema and not by the order of the items' occurrence in the LIST array or NAME-LIST.
- o Unreferenced data items are filled with binary zeroes or ASCII blanks.

DETAIL DATA SETS

When adding entries to detail data sets the following rules apply:

- The data set must have free space for the entry.
- All key and sort items defined for the entry must be referenced in the LIST array or NAME-LIST.
- Each related manual master data set must contain a matching entry for the corresponding key item value. If any automatic master does not have a matching entry, it must have space to add one. This addition occurs automatically.
- Sort items must be assigned a value, or an error code of 102 will be returned.
- The order of data item values in the new entry is determined by the set definition in the schema and not by the order of the items' occurrence in the LIST array or NAME-LIST.
- Unreferenced data items are filled with binary zeroes or ASCII blanks.
- The new entry is linked into one chain for each key item or path defined according to the key item value. It is linked to the end of the chain if the chain is unsorted; otherwise, it is placed in the chain according to the value of its sort item for that chain.

STATUS CODES:

The following are status codes returned from DBPUT. For more information about these codes, see Appendix B.

CODE	MEANING
100	Illegal data set reference
101	Illegal item list
102	Improper or missing search or sort item in item list
103	Illegal database parameter or database not open
104	Data base not enabled for DBPUT (open in mode 8)
105	Detail data set is full
106	Master data set is full
107	A manual master does not contain a record with a key item value (detail data set only)
108	Request directed at an automatic master
110	Master record with the key item value already exists (Manual master only)
115	Invalid DBPUT mode
118	Data set not write enabled for user's access level

154	Data base corrupt, bad chain pointers
159	Data base was opened in mode 1 and has not been locked to the user
160	IMAGE data structures are corrupt
162	Missing parameter in DBPUT call

In addition, BASIC may return the following status codes:

CODE	MEANING
302	Too many names in NAME-LIST (more than ten)
303	Invalid name in NAME-LIST
304	The type or length of a variable in the VALUE-LIST does not match that of its corresponding item in the NAME-LIST
305	Variable missing in VALUE-LIST (more names in NAME-LIST than variables in VALUE-LIST)

TEXT DISCUSSION:

Page 4-11

PARAMETER DISCUSSION:

IBASE	Page 4-17
ID	Page 4-18
IMODE	Page 4-18
ISTAT	Page 4-18
LIST	Page 4-19
NAME-LIST	Page 4-20
IBUF	Page 4-18
VALUE-LIST	Page 4-20

EXAMPLES:

FORTRAN	Page 5-9
PASCAL	Page 5-35
Assembler	Page 5-39
BASIC	Page 5-47

DBUNL

DBUNL(ibase,id,imode,istat)

Relinquishes the temporary exclusive control a database acquired by a previous call to DBLCK. DBUNL unlocks the root file and disables the calling program's database write access until another DBLCK call is made. The database must be open in mode 1 or the call is ignored. Redundant calls are also ignored.

PARAMETERS:

ibase	is the array used for the IBASE parameter when opening the database. The array must not have been altered since issuing the call to DBOPN.
id	is unused. Use a dummy variable or any ID array used for a previous call.
imode	must be an integer equal to 1.
istat	is an array in which IMAGE returns status information. If the operation was successful, the first word or element of the array is set to zero.

STATUS CODES:

The following are the possible status codes returned from DBUNL. For more information about these codes, see Appendix B.

CODE	MEANING
103	Illegal database parameter or database not open
115	Illegal DBUNL mode
137	Illegal resource number usage (resource number released or written over)

TEXT DISCUSSION:

Page 4-14

PARAMETER DISCUSSION:

IBASE	Page 4-17
ISTAT	Page 4-18

EXAMPLES:

FORTRAN	Page 5-14
PASCAL	Page 5-35
Assembler	Page 5-40
BASIC	Page 5-50

DBUPD

$$\text{DBUPD(ibase,id,imode,istat,}\ \left\{\begin{array}{c}\text{list}\\ \text{name-list}\end{array}\right\}\ ,\ \left\{\begin{array}{c}\text{ibuf}\\ \text{value-list}\end{array}\right\}\ )$$

Modifies values of data items in the entry residing at the current record address of a specified data set. Key and sort items cannot be modified. The database must be open in mode 1 or 3. If open in mode 1, the database must have been locked with a DBLCK call.

PARAMETERS:

ibase is the array used as the IBASE parameter when the database was opened. This array must not have been altered since issuing the call to DBOPN.

id is an array containing the data set name, or if FORTRAN or Assembly Language, an integer variable containing the data set number.

imode must be an integer equal to 1.

istat is an array in which IMAGE returns status information to the user. If the operation was successful, the contents of the array will be as follows:

Word	Contents
1	Zero
2	Length of the values in IBUF or VALUE-LIST

list OR name-list is an array containing a set of data item names, or if FORTRAN, PASCAL, or Assembly Language, data item numbers, to be updated. NAME-LIST, used in BASIC calls, cannot contain a special construct.

ibuf OR value-list contains the values to be assigned to those data items listed in LIST or NAME-LIST. If in FORTRAN, PASCAL, or Assembly language, the values are concatenated. If in BASIC, each variable must correspond in type and length with its corresponding item in NAME-LIST. In any language, the items must appear in the same order as the corresponding items in LIST or NAME-LIST.

STATUS CODES:

The following are the status codes from DBUPD. For more information regarding these codes, see Appendix B.

CODE	MEANING
100	Illegal data set reference
101	Illegal item list
103	Illegal database parameter or database not open
104	Data base not enabled for updating (open in mode 8)
108	Request directed at an automatic master
112	Attempt to alter the value of a key, sort, or nonwritable item
114	Record accessed is empty
115	Invalid DBUPD mode
157	No current record
159	Data base was opened in mode 1 and has not been locked to the user.
160	IMAGE data structures are corrupt
162	Missing parameter in DBUPD call

In addition, BASIC may return the following error codes:

CODE	MEANING
302	Too many names in NAME-LIST (more than ten)
303	Invalid name in NAME-LIST
304	The type or length of a variable in the VALUE-LIST does not match that of its corresponding item in the NAME-LIST
305	Variable missing in VALUE-LIST (more names in NAME-LIST than variables in VALUE-LIST)

DBUPD

TEXT DISCUSSION:

Page 4-10

PARAMETER DISCUSSION:

IBASE	Page 4-17
ID	Page 4-18
IMODE	Page 4-18
ISTAT	Page 4-18
LIST	Page 4-19
NAME-LIST	Page 4-20
IBUF	Page 4-18
VALUE-LIST	Page 4-20

EXAMPLES:

FORTRAN	Page 5-11
PASCAL	Page 5-35
Assembler	Page 5-39
BASIC	Page 5-48

Chapter 5
Host Language Examples

This chapter is divided into three separate discussions, each covering the use of IMAGE with a specific programming language: FORTRAN, PASCAL, Assembly Language and BASIC (note that BASIC examples are not applicable to 92073A users). The examples in each language are designed to illustrate simply and directly the way IMAGE subroutines are called.

A knowledge of the programming language is assumed. Questions about the language itself can be answered by consulting the appropriate language manual. All examples assume that the program is being run from a session terminal and therefore address LU 1 for reading and writing.

All examples presented in this chapter perform operations on the QA database. Chapters 1 and 2 should be consulted if questions about the database structure arise in relation to the example.

FORTRAN Examples

In the FORTRAN examples that follow, all variables which expect alphanumeric data are typed integer at the beginning of the program. Where required, arrays are inititalized with data statements. Statement 1600, though not shown, contains code to close the database. All FORTRAN examples, with the exception of the example of a directed read, are taken from one of three programs - PART, PROJ or TRANS. Under the discussion of a particular subroutine, only those lines of program that are applicable are shown. The declarative section, if shown, will contain only those items which pertain to the example. However, the programs in their entirety are shown at the end of this section.

Open Database

```
FTN4,L
      PROGRAM TRANS
      INTEGER LEVEL(3),QA(6),STAT(10)
      DATA QA/2H  ,2HQA,2H:1,2H00,2H:4,2H3;/
      DATA MODE1/1/
      DATA MODE7/7/
      WRITE(1,100)
100   FORMAT("PLEASE ENTER PROPER SECURITY CODE.")
      READ(1,*)ISEC
      IF(ISEC.EQ.ICODE)GO TO 400
      WRITE(1,300)
300   FORMAT("BAD SECURITY CODE - GOOD-BYE!")
      GO TO 9900
400   WRITE(1,500)
500   FORMAT("PLEASE ENTER YOUR LEVEL.")
      READ(1,600)(LEVEL(I),I=1,3)
600   FORMAT(3A2)
      CALL DBOPN(QA,LEVEL,MODE1,STAT)
      IF(STAT(1).EQ.0)GO TO 710
      IF(STAT(1).NE.153)GO TO 750
      WRITE(1,700)
700   FORMAT("YOUR LEVEL DOESN'T ALLOW ACCESS TO THE DATABASE."/
     1"  GOOD-BYE!")
      GO TO 9900
```

In this example, the QA database is opened after the user enters the security code and level. Later, a subroutine will be used to determine if the user's level is sufficient to enter a transaction.

The database is opened in mode 1, which allows read/write access concurrent with other programs opened in mode 1. The sample program PART (see page 5-18) which only reads the database opens in mode 8, since write access is not necessary.

The first two characters of the array containing the database name are blanks, which indicates that this database is at a local node. The remainder of the QA array contains the full database FMP file namr, including file security code and cartridge.

Request Database Information

```
FTN4,L
      SUBROUTINE INFO(DBASE,BASDM,DSET,ARRAY,FLAG)
      INTEGER DBASE(BASDM)
      INTEGER DSET(3)
      INTEGER BASDM
      INTEGER BUF(256)
      INTEGER STAT(10)
      INTEGER BUF2(13)
      INTEGER FLAG
      INTEGER ARRAY(10,6)
      INTEGER CUM
      DO 100 I=1,10
      DO 100 J=1,6
      ARRAY(I,J)=2H
100   CONTINUE
C
C           CALL DBINF TO GET A LIST OF ITEM NUMBERS AND
C           WHETHER ACCESS IS READ ONLY OR READ/WRITE
C
      CALL DBINF(DBASE,DSET,104,STAT,BUF)
      IF(STAT(1).NE.0)GO TO 300
      N=BUF(1)+1
      IF(N.GT.10)GO TO 400
      CUM=1
      DO 200 I=2,N
      M=I-1
C
            CALL DBINF TO GET ITEM NAME AND LENGTH.
C           IF ITEM TYPE IS X, THE LENGTH GIVEN IS IN
C           CHARACTERS, NOT WORDS.
C
      CALL DBINF(DBASE,IABS(BUF(I)),102,STAT,BUF2)
      IF(STAT(1).NE.0)GO TO 300
      ARRAY(M,1)=BUF2(1)
      ARRAY(M,2)=BUF2(2)
      ARRAY(M,3)=BUF2(3)
      ARRAY(M,4)=BUF2(10)*BUF2(11)
      IF(BUF2(9).EQ.2HX )ARRAY(M,4)=ARRAY(M,4)/2
      ARRAY(M,5)=CUM
      IF(BUF(I).LT.0)ARRAY(M,6)=2H$$
      CUM=CUM+ARRAY(M,4)
200   CONTINUE
      FLAG=0
      GO TO 900
C
C          IF THERE WAS AN ERROR IN THE DBINF CALL, THE FLAG
```

```
C         VALUE RETURNED IS THE NEGATIVE OF THE IMAGE ERROR CODE
C
300    FLAG=-STAT(1)
       GO TO 900
C
C         IF THE TABLE IS NOT LARGE ENOUGH TO HANDLE ALL THE
C         ITEMS, A FLAG VALUE OF +1 IS RETURNED
C
400    FLAG=+1
900    CONTINUE
       RETURN
       END
       END$
```

This subroutine is used to build an array containing the item name, beginning word position, item length in words, and a flag to determine if the user has read only or read/write access. Two calls are made: a mode 104 to determine the items in the set to which the user has access and a mode 102 to determine the item's name and length.

Read Entry (Forward Chain)

```
FTN4,L
      PROGRAM PART
      INTEGER QA(6),LEVEL(3),STAT(10)
      INTEGER PART(4),PARTN(3),ASET(3),FSET(3),BUF(23)
      DATA QA/2H  ,2HQA,2H:1,2H00,2H:4,2H3;/
      DATA FSET/2HFA,2HIL,2H  /
      DATA PARTN/2HPA,2HRT,2H# /
      DATA LALL/2H@ /
      DATA MODE1/1/
      DATA MODE2/2/
      DATA MODE4/4/
      DATA MODE5/5/
      DATA MODE7/7/
      DATA MODE8/8/
                          .
                          .
                          .
        code to open database and determine function
                          .
                          .
                          .
500   WRITE(1,510)
510   FORMAT("WHAT PART ARE YOU INTERESTED IN?"/
     1"(EX ENDS THIS FUNCTION)")
      READ(1,520)PART
520   FORMAT(4A2)
      IF(PART(1).EQ.2HEX)GO TO 595
      CALL DBFND(QA,FSET,MODE1,STAT,PARTN,PART)
      IF(STAT(1).EQ.0)GO TO 570
      IF(STAT(1).EQ.156)GO TO 550
      IF(STAT(1).NE.107)GO TO 535
      WRITE(1,530)
530   FORMAT("NO RECORD FOR THAT PART EXISTS")
      GO TO 500
535   WRITE(1,540)STAT(1)
540   FORMAT("PROGRAM ERROR ON FIND = ",I5)
      GO TO 1600
550   WRITE(1,560)
560   FORMAT("NO HISTORY EXISTS FOR THIS PART ")
```

```
      GO TO 500
570   WRITE(1,572)(PART(I),I=1,4)
572   FORMAT("PART FAILURE HISTORY FOR PART NUMBER ",4A2,/
     1"    ASSY#     DATE    PART#    CODE REF",/)
575   CALL DBGET(QA,FSET,MODE5,STAT,LALL,BUF,ARG)
      IF(STAT(1).EQ.0)GO TO 580
      IF(STAT(1).EQ.155)GO TO 500
      WRITE(1,579)STAT(1)
579   FORMAT("PROGRAM ERROR ON GET =",I5)
      GO TO 1600
580   WRITE(1,590)(BUF(I),I=1,16)
590   FORMAT(1X,5A2,1X,3A2,1X,4A2,1X,2A2,1X,2A2)
      GO TO 575
```

In order to do a chained read, a call to DBFND must be done to properly initialize the chain pointers. If the status returned is 107, there is no master entry for that key value. If the status code returned is 156, the master entry exists but there are no detail records in the chain.

Once the call to DBFND has completed successfully, the chain can be read with repeated calls to DBGET in mode 5. A status code returned of 155 indicates that there are no more records in the chain and the function is complete.

In this example, the special construct "@ " is used to specify the list of items desired. This construct will return all items in the entry.

Read Entry (Serially)

```
FTN4,L
      PROGRAM PART
      INTEGER QA(6),LEVEL(3),STAT(10)
      DATA QA/2H  ,2HQA,2H:1,2H00,2H:4,2H3;/
      DATA LALL/2H@ /
      DATA ASET/2HAS,2HMB,2HY /
      DATA MODE1/1/
      DATA MODE2/2/
      DATA MODE4/4/
      DATA MODE5/5/
      DATA MODE7/7/
      DATA MODE8/8/
                              .
                              .
                              .
            code to open database and determine function
                              .
                              .
                              .
700   CONTINUE
      ARG = 0
      CALL DBGET(QA,ASET,MODE4,STAT,LALL,BUF,ARG)
      WRITE(1,705)
705   FORMAT("ALL ASSEMBLIES FAILURE COUNTS"//
     1"     ASSY#     FAILS"/)
710   CALL DBGET(QA,ASET,MODE2,STAT,LALL,BUF,ARG)
      IF(STAT(1).EQ.0)GO TO 740
      IF(STAT(1).EQ.12)GO TO 780
      WRITE(1,720)STAT(1)
720   FORMAT("ERROR ON GET = ",I5)
      GO TO 1600
740   WRITE(1,750)(BUF(I),I=1,6)
750   FORMAT(1X,5A2,2X,I5)
      GO TO 710
780   CONTINUE
```

Serial reads are performed with successive calls to DBGET in mode 2. Before beginning the serial reads, a call is made to DBGET in mode 4 (directed read) with an ARG value of zero to reset the file position pointer to the beginning of the file. A status return of 12 from the serial reads indicates that the end of the file has been reached.

In this example, the special construct "@ " is used as the item list to return all items in the entry.

Read Entry (Directed)

```
FTN4,L
      PROGRAM DIRCT
      INTEGER QA(6),LEVEL(3),STAT(10)
      INTEGER FSET(3),BUF(23),PARTN(3),PART(4),ASET(3)
      INTEGER LIST(9), IREC(2)
      DATA QA/2H  ,2HQA,2H:1,2H00,2H:4,2H3;/
      DATA FSET/2HFA,2HIL,2H  /
      DATA LIST/2HAS,2HSY,2H#,,2HPA,2HRT,2H#,,2HEM,2HP#,2H; /
      DATA MODE4/4/
                                  .
                                  .
                                  .
                  code to open the database
                                  .
                                  .
                                  .
500   WRITE(1,510)
510   FORMAT("WHAT ENTRY DO YOU WANT?"/"0 ENDS THIS PROGRAM.")
      READ(1,*)IREC(2)
      IF(IREC(2).EQ.0)GO TO 1600
      CALL DBGET(QA,FSET,MODE4,STAT,LIST,BUF,IREC)
      IF(STAT(1).EQ.111)GO TO 530
      IF(STAT(1).EQ.114)GO TO 540
      WRITE(1,520)STAT(1)
520   FORMAT("SYSTEM ERROR = ",I5," ON DATABASE GET.")
      GO TO 1600
530   WRITE(1,535)
535   FORMAT("ILLEGAL RECORD NUMBER")
      GO TO 500
540   WRITE(1,545)
545   FORMAT("RECORD IS EMPTY")
      GO TO 500
```

The user is prompted for a relative record number and a call to DBGET is made in mode 4 to access that record. If the record number is illegal (doesn't exist) a status code of 111 is returned. If the record accessed is empty, a status code of 114 is returned.

In this example, a data item name list is used to obtain only the assembly, part and employee numbers. If a data item number list was to be used, the data statement for LIST would be

```
     DATA LIST/3,1,3,7/
```

because assembly number is item 1, part number is item 3 and employee number is item 7.

For a directed read, the last parameter, relative record number, is a doubleword integer.

Read Entry (Keyed)

```
FTN4,L
      PROGRAM PROJ
      INTEGER QA(6),LEVEL(3),STAT(10)
      INTEGER PROJ(3),DPROJ(3),ZERO,DATE(3)
      INTEGER BUF(11)
      DATA QA/2H  ,2HQA,2H:1,2H00,2H:4,2H3;/
      DATA PROJ/2HPR,2HOJ,2H  /
      DATA ZERO/0/
      DATA MODE1/1/
      DATA MODE2/2/
      DATA MODE7/7/
                                   .
                                   .
                                   .
        code to open database and determine function
                                   .
                                   .
                                   .
600   WRITE(1,605)
605   FORMAT("WHAT IS THE PROJECT NUMBER TO BE DELETED?"
     1"(EX ENDS THIS FUNCTION.)")
      READ(1,610)(DPROJ(I),I=1,3)
610   FORMAT(3A2)
      IF(DPROJ(1).EQ.2HEX)GO TO 660
      CALL DBLCK(QA,ID,MODE1,STAT)
      IF(STAT(1).EQ.0)GO TO 620
613   WRITE(1,615)
615   FORMAT("SYSTEM ERROR = ",I5," ON LOCK."/
     1"CALL A SYSTEMS ANALYST.")
      GO TO 1600
620   CALL DBGET(QA,PROJ,MODE7,STAT,ZERO,BUF,DPROJ)
      IF(STAT(1).EQ.0)GO TO 635
      IF(STAT(1).NE.107)GO TO 625
      WRITE(1,621)
621   FORMAT("THIS PROJECT NUMBER DOES NOT EXIST.")
622   CALL DBUNL(QA,ID,MODE1,STAT)
      IF(STAT(1).EQ.0)GO TO 600
      GO TO 556
625   WRITE(1,626)STAT(1)
626   FORMAT("SYSTEM ERROR = ",I5," ON GET."/
     1"CALL A SYSTEMS ANALYST.")
      GO TO 1600
```

Here a call to DBGET is made with a mode of 7 which performs a keyed read. In this case, the last parameter is an integer array, as the key is a character key.

If an entry with the desired key value does not exist, a status code of 107 is returned.

In this DBGET call, another special construct is used. The item list is a zero which means to position to the record, but do not read it. This construct is used in this example because the record is to be deleted after being located and no information from the record is desired.

Add An Entry

```
FTN4,L
      PROGRAM TRANS
      INTEGER LEVEL(3),QA(6),STAT(10)
      INTEGER BUF(23),BUF2(6)
      INTEGER FAIL(3),ASMBY(3),ASSYN(5)
      DATA QA/2H  ,2HQA,2H:1,2H00,2H:4,2H3;/
      DATA FAIL/2HFA,2HIL,2H  /
      DATA ASMBY/2HAS,2HMB,2HY /
      DATA LIST/2H@ /
      DATA MODE1/1/
      DATA MODE7/7/
                                    .
                                    .
                                    .
               code to open database and determine
             beginning and ending character positions
                    in the entry of each item
                                    .
                                    .
                                    .
800   WRITE(1,900)
900   FORMAT("GET READY TO ENTER A TRANSACTION."/
     1"EN ENDS THIS PROGRAM."/
     2"ENTER ASSEMBLY NUMBER.")
      READ(1,1000)(BUF(I),I=A1,A2)
1000  FORMAT(5A2)
      J = 1
      DO 1005 I=A1,A2
      ASSYN(J) = BUF(I)
      J = J + 1
1005  CONTINUE
      IF(BUF(A1).EQ.2HEN)GO TO 1600
      WRITE(1,1010)
1010  FORMAT("ENTER DATE AS YYMMDD")
```

```
      READ(1,1020)(BUF(I),I=D1,D2)
1020  FORMAT(3A2)
      WRITE(1,1030)
1030  FORMAT("ENTER PART NUMBER")
      READ(1,1040)(BUF(I),I=P1,P2)
1040  FORMAT(4A2)
      WRITE(1,1050)
1050  FORMAT("ENTER FAILED CODE")
      READ(1,1060)(BUF(I),I=F1,F2)
1060  FORMAT(2A2)
      WRITE(1,1070)
1070  FORMAT("ENTER REFERENCE DESIGNATOR")
      READ(1,1080)(BUF(I),I=R1,R2)
1080  FORMAT(2A2)
      WRITE(1,1090)
1090  FORMAT("ENTER POINT VALUES (0 OR 1)")
      READ(1,1100)(BUF(I),I=PT1,PT2)
1100  FORMAT(4A2)
      WRITE(1,1110)
1110  FORMAT("ENTER EMPLOYEE NUMBER")
      READ(1,1120)(BUF(I),I=E1,E2)
1120  FORMAT(3A2)
      CALL DBLCK(QA,ID,MODE1,STAT)
      IF(STAT(1).NE.0)GO TO 1500
      CALL DBPUT(QA,FAIL,MODE1,STAT,LIST,BUF)
      IF(STAT(1).NE.0)GO TO 1520
1200  CALL DBUNL(QA,ID,MODE1,STAT)
      IF(STAT(1).EQ.0)GO TO 800
      WRITE(1,1300)STAT(1)
1300  FORMAT("SYSTEM ERROR = ",I5,"ON UNLOCK."/
     1"CALL A SYSTEMS ANALYST.")
      GO TO 1600
1500  WRITE(1,1505)STAT(1)
1505  FORMAT("SYSTEM ERROR = ",I5," ON LOCK."/
     1"CALL A SYSTEMS ANALYST.")
      GO TO 1600
1520  IF(STAT(1).EQ.107)GO TO 1527
      WRITE(1,1525)STAT(1)
1525  FORMAT("SYSTEM ERROR = ",I5," ON PUT."/
     1"CALL A SYSTEMS ANALYST.")
      GO TO 1600
1527  WRITE(1,1528)
1528  FORMAT("ILLEGAL PART NUMBER.")
      GO TO 1200
                            .
                            .
                            .
                code to close database
                            .
                            .
                            .
```

In code not shown, calls have been made to DBINF to determine the beginning and ending positions of each item in the entry. The user is prompted for values for each item and his responses are placed in the buffer based on the position information determined previously.

In the call to DBPUT, the special construct of "@ " is used for the list array. This indicates that all items are to be placed in the entry.

The database must be locked before the call to DBPUT because the open had been in mode 1. If no entry with the supplied key value exists in the associated master data set, a status code of 107 is returned. Before going to prompt the user for another transaction, the database must be unlocked. In the case of an error which causes a jump to statement 1600 to close the database, an unlock is not needed since DBCLS will unlock a locked database.

Update An Entry

```
FTN4,L
      PROGRAM TRANS
      INTEGER LEVEL(3),QA(6),STAT(10)
      INTEGER BUF(23),BUF2(6)
      INTEGER FAIL(3),ASMBY(3),ASSYN(5)
      DATA QA/2H  ,2HQA,2H:1,2H00,2H:4,2H3;/
      DATA FAIL/2HFA,2HIL,2H  /
      DATA ASMBY/2HAS,2HMB,2HY /
      DATA LIST/2H@ /
      DATA MODE1/1/
      DATA MODE7/7/
                             .
                             .
                             .
              code to open the database and
               determine the beginning and
                ending character positions
                in the entry of each item
                             .
                             .
                             .
      CALL DBLCK(QA,ID,MODE1,STAT)
      IF(STAT(1).NE.0)GO TO 1500
      CALL DBGET(QA,ASMBY,MODE7,STAT,LIST,BUF2,ASSYN)
      IF(STAT(1).NE.0)GO TO 1510
      BUF2(FN1) = BUF2(FN1) + 1
      CALL DBUPD(QA,ASMBY,MODE1,STAT,LIST,BUF2)
      IF(STAT(1).NE.0)GO TO 1530
1200  CALL DBUNL(QA,ID,MODE1,STAT)
      IF(STAT(1).EQ.0)GO TO 800
      WRITE(1,1300)STAT(1)
1300  FORMAT("SYSTEM ERROR = ",I5,"ON UNLOCK."/
     1"CALL A SYSTEMS ANALYST.")
      GO TO 1600
1500  WRITE(1,1505)STAT(1)
1505  FORMAT("SYSTEM ERROR = ",I5," ON LOCK."/
     1"CALL A SYSTEMS ANALYST.")
      GO TO 1600
1510  IF(STAT(1).EQ.107)GO TO 1517
      WRITE(1,1515)STAT(1)
1515  FORMAT("SYSTEM ERROR = ",I5," ON GET."/
     1"CALL A SYSTEMS ANALYST.")
      GO TO 1600
1517  WRITE(1,1518)
1518  FORMAT("ILLEGAL ASSEMBLY NUMBER.")
      GO TO 1200
1530  WRITE(1,1535)STAT(1)
1535  FORMAT("SYSTEM ERROR = ",I5," ON UPDATE."/
     1"CALL A SYSTEMS ANALYST.")
                             .
                             .
                             .
              code to close the database
                             .
                             .
                             .
```

A database lock is done before the call to DBGET to insure that the entry is not modified between this program's get and update. After the entry has been updated, the database is unlocked and the user is prompted for another entry.

Delete An Entry

```
FTN4,L
      PROGRAM PROJ
      INTEGER QA(6),LEVEL(3),STAT(10)
      INTEGER PROJ(3),DPROJ(3),ZERO,DATE(3)
      INTEGER BUF(11)
      DATA QA/2H  ,2HQA,2H:1,2H00,2H:4,2H3;/
      DATA PROJ/2HPR,2HOJ,2H  /
      DATA ZERO/0/
      DATA MODE1/1/
      DATA MODE2/2/
      DATA MODE7/7/
600   WRITE(1,605)
605   FORMAT("WHAT IS THE PROJECT NUMBER TO BE DELETED?"
     1"(EX ENDS THIS FUNCTION.)")
      READ(1,610)(DPROJ(I),I=1,3)
```

```
610   FORMAT(3A2)
      IF(DPROJ(1).EQ.2HEX)GO TO 660
      CALL DBLCK(QA,ID,MODE1,STAT)
      IF(STAT(1).EQ.0)GO TO 620
613   WRITE(1,615)
615   FORMAT("SYSTEM ERROR = ",I5," ON LOCK."/
     1"CALL A SYSTEMS ANALYST.")
      GO TO 1600
620   CALL DBGET(QA,PROJ,MODE7,STAT,ZERO,BUF,DPROJ)
      IF(STAT(1).EQ.0)GO TO 635
      IF(STAT(1).NE.107)GO TO 625
      WRITE(1,621)
621   FORMAT("THIS PROJECT NUMBER DOES NOT EXIST.")
622   CALL DBUNL(QA,ID,MODE1,STAT)
      IF(STAT(1).EQ.0)GO TO 600
      GO TO 556
625   WRITE(1,626)STAT(1)
626   FORMAT("SYSTEM ERROR = ",I5," ON GET."/
     1"CALL A SYSTEMS ANALYST.")
      GO TO 1600
635   CALL DBDEL(QA,PROJ,MODE1,STAT)
      IF(STAT(1).EQ.0)GO TO 622
      IF(STAT(1).EQ.113)GO TO 642
      WRITE(1,640)STAT(1)
640   FORMAT("SYSTEM ERROR = ",I5," ON DELETE."/
     1"CALL A SYSTEMS ANALYST.")
      GO TO 1600
642   WRITE(1,645)
645   FORMAT("TRANSACTIONS STILL EXIST FOR THIS PROJECT NUMBER."/
     1"IT CANNOT BE DELETED.")
      GO TO 622
```

The steps in deleting an entry from a database opened in mode 1 (shared mode) are lock, get, delete and unlock. If the database is opened in exclusive mode (mode 3) the lock and unlock steps are omitted.

If the entry to be deleted is in a manual master data set and there are still entries in a detail data set linked to the master entry, a status code of 113 is returned and no deletion is made.

Lock And Unlock Database

```
FTN4,L
      PROGRAM PROJ
      INTEGER QA(6),LEVEL(3),STAT(10)
      INTEGER PROJ(3),DPROJ(3),ZERO,DATE(3)
      INTEGER BUF(11)
      DATA QA/2H  ,2HQA,2H:1,2H00,2H:4,2H3;/
      DATA PROJ/2HPR,2HOJ,2H  /
      DATA MODE1/1/
      DATA MODE2/2/
      DATA MODE7/7/
                          •
                          •
              code to open the database
                          •
                          •
                          •
      WRITE(1,705)
705   FORMAT("WHAT PROJECT NUMBER DO YOU WANT CLOSED?"/
     1"(EX ENDS THIS FUNCTION.)")
      READ(1,710)(DPROJ(I),I=1,3)
710   FORMAT(3A2)
      IF(DPROJ(1).EQ.2HEX)GO TO 660
      WRITE(1,715)
715   FORMAT("WHAT IS THE CLOSING DATE?")
      READ(1,720)(DATE(I),I=1,3)
720   FORMAT(3A2)
      CALL DBLCK(QA,ID,MODE1,STAT)
      IF(STAT(1).NE.0)GO TO 613
      CALL DBGET(QA,PROJ,MODE7,STAT,LIST,BUF,DPROJ)
      IF(STAT(1).EQ.0)GO TO 730
      IF(STAT(1).NE.107)GO TO 625
      WRITE(1,725)
725   FORMAT("THIS PROJECT NUMBER DOESN'T EXIST.")
      CALL DBUNL(QA,ID,MODE1,STAT)
      IF(STAT(1).EQ.0)GO TO 700
      GO TO 556
```

If a database is opened in shared mode (mode 1), the database must be locked before updating, adding or deleting entries. The above example cans DBLCK before a DBGET to insure the integrity of that entry prior to updating it.

If the database is opened in mode 3 (exclusive read/write), no error will occur if an attempt to lock the database is made, but no action will be taken. A database opened in mode 8 (read only) cannot be locked and an attempt to do so will result in a status code of 134.

In all examples shown, the lock requests are made in mode 1 which is lock with wait. No return will be made to the user until the requested operation is complete. An alternative would be to request lock without wait (mode 2) in which case return is immediate with an indication of whether or not the requested operation took place.

A database need not be unlocked to be closed. If a request is made to close a locked database, an attempt to unlock will be made at that time. If the unlock is unsuccessful, the close request will return a status code of 135.

Close A Data Set

```
FTN4,L
      PROGRAM PART
      INTEGER QA(6),LEVEL(3),STAT(10)
      INTEGER PART(4),PARTN(3),ASET(3),FSET(3),BUF(23)
      DATA QA/2H  ,2HQA,2H:1,2H00,2H:4,2H3;/
      DATA FSET/2HFA,2HIL,2H  /
      DATA MODE1/1/
      DATA MODE2/2/
      DATA MODE4/4/
      DATA MODE5/5/
      DATA MODE7/7/
      DATA MODE8/8/
                              .
                              .
                              .
             code to open database and determine function
                              .
                              .
                              .
500   WRITE(1,510)
510   FORMAT("WHAT PART ARE YOU INTERESTED IN?"/
     1"(EX ENDS THIS FUNCTION)")
      READ(1,520)PART
520   FORMAT(4A2)
      IF(PART(1).EQ.2HEX)GO TO 595
595   CONTINUE
      CALL DBCLS(QA,FSET,MODE2,STAT)
      IF(STAT(1).EQ.0)GO TO 345
      WRITE(1,597)STAT(1)
597   FORMAT("ERROR ON DATA SET CLOSE = ",I5)
      GO TO 1600
```

In this example, when the user is finished with his transaction, a call to DBCLS in mode 2 is made to close a particular data set. The data set is not specifically opened by the user, but is opened by IMAGE when the data set is first accessed. Data set closes are optional and if not done previously will be done when the database is closed.

Close Database

```
FTN4,L
      PROGRAM PART
      INTEGER QA(6),LEVEL(3),STAT(10)
      DATA QA/2H  ,2HQA,2H:1,2H00,2H:4,2H3;/
      DATA MODE1/1/
      DATA MODE2/2/
      DATA MODE4/4/
      DATA MODE5/5/
      DATA MODE7/7/
      DATA MODE8/8/
                             .
                             .
                             .
                code to operate on database
                             .
                             .
                             .
1600  CONTINUE
      CALL DBCLS(QA,ID,MODE1,STAT)
      IF(STAT(1).EQ.0)GO TO 1800
      WRITE(1,1700)STAT(1)
1700  FORMAT("ERROR ON DATABASE CLOSE = ",I5)
1800  CONTINUE
      END
```

All exits from the program should be through this one section of code to close the database. An error in closing the data base should be reported to give some indication of possible problems with the database.

Sample FORTRAN Programs

The following are examples of FORTRAN programs accessing the QA data base. There are three programs - PART, PROJ and TRANS, and two subroutines - INFO and OFSET.

The PART program allows the user to choose one of three options - failure count for all assemblies, failure count for a specific assembly, or failure history for a specific part. PROJ allows the addition, deletion or closing of a project number. TRANS prompts the user for failure information in order to make an entry in the failure data set.

```
FTN4,L
      PROGRAM PART
      INTEGER QA(7),LEVEL(3),STAT(10)
      INTEGER PART(4),PARTN(3),ASET(3),FSET(3),BUF(23)
      INTEGER PSET(3),ASMB(5)
      INTEGER ARY1(10,6),ARY2(10,6),FLAG,FUNC
      DATA QA/2H  ,2HQA,2HDB,2H:1,2H00,2H:4,2H3;/
      DATA FSET/2HFA,2HIL,2H  /
      DATA PARTN/2HPA,2HRT,2H# /
      DATA LALL/2H@ /
      DATA ASET/2HAS,2HMB,2HY /
      DATA MODE1/1/
      DATA MODE2/2/
      DATA MODE4/4/
      DATA MODE5/5/
      DATA MODE7/7/
      DATA MODE8/8/
C
C     THE USER IS PROMPTED FOR AN ACCESS LEVEL CODE WORD
C     AND AN ATTEMPT IS MADE TO OPEN THE QA DATABASE
C     USING THAT WORD.  IF THE USER HAS NO ACCESS TO ANY ITEMS
C     IN THE DATABASE, A MESSAGE IS PRINTED AND THE PROGRAM
C     TERMINATES.  IF THE OPEN IS SUCCESSFUL, THE PROGRAM
C     CONTINUES.
C
      WRITE(1,100)
100   FORMAT("QA REPORT PROGRAM"/"ENTER LEVEL")
      READ(1,105)(LEVEL(I),I=1,3)
105   FORMAT(3A2)
      CALL DBOPN(QA,LEVEL,MODE8,STAT)
      IF(STAT(1).EQ.0)GO TO 335
      IF(STAT(1).EQ.153)GO TO 330
      WRITE(1,325)STAT(1)
325   FORMAT("SYSTEM ERROR = ",I5," ON DB OPEN"/
     1"CALL A SYSTEMS ANALYST.")
      GO TO 1800
330   WRITE(1,331)
331   FORMAT("YOUR LEVEL DOESN'T ALLOW SUFFICIENT ACCESS FOR ANY TR
     1ANSACTION.")
      GO TO 1800
C
C     THE USER WILL BE ASKED TO PICK A FUNCTION.  THERE ARE
C     FOUR ACCEPTABLE ANSWERS.  THE USER HAS THREE CHANCES
C     TO ENTER AN ACCEPTABLE RESPONSE.  IF HE CANNOT, THE
C     PROGRAM IS TERMINATED.  IF THE USER RESPONSE IS ONE
C     OF THE PROPER FUNCTIONS, THE PROGRAM BRANCHES TO PROCESS
C     THAT FUNCTION.
C
335   CALL INFO(QA,7,FSET,ARY1,FLAG)
      IF(FLAG.EQ.0)GO TO 340
      IF(FLAG.EQ.1)GO TO 337
      WRITE(1,336)FLAG
336   FORMAT("SYSTEM ERROR = ",I5," ON DB INFO CALL."/
     1"CALL A SYSTEMS ANALYST.")
      GO TO 1600
337   WRITE(1,338)
338   FORMAT("TOO MANY ITEMS IN DATA SET FOR TABLE."/
     1"CALL A SYSTEMS ANALYST.")
      GO TO 1600
340   CALL INFO(QA,7,ASET,ARY2,FLAG)
      IF(FLAG.EQ.0)GO TO 345
      IF(FLAG.EQ.1)GO TO 337
      WRITE(1,336)FLAG
      GO TO 1600
345   ICNT = 0
346   WRITE(1,347)
347   FORMAT("CHOOSE A FUNCTION."/
     1"PH = PART HISTORY"/"AF = ASSEMBLY FAILURE COUNT"/
     2"AA = ALL ASSEMBLIES FAILURE COUNT"/"EP = END PROGRAM")
      READ(1,348)FUNC
348   FORMAT(A2)
      IF(FUNC.EQ.2HPH)GO TO 500
      IF(FUNC.EQ.2HAF)GO TO 600
      IF(FUNC.EQ.2HAA)GO TO 700
      IF(FUNC.EQ.2HEP)GO TO 1600
350   ICNT=ICNT+1
      IF(ICNT.EQ.3)GO TO 800
      WRITE(1,400)
400   FORMAT("FUNCTION INPUT ERROR - TRY AGAIN!")
      GO TO 346
C
C     THIS SECTION PRODUCES A FAILURE HISTORY FOR A PART
C     NUMBER SUBMITTED BY THE USER.  THE USER IS PROMPTED
C     FOR THE PART NUMBER AND THE CHAIN FOR THAT PART NUMBER
C     IS PRINTED OUT.  IF THERE IS NOT AN ENTRY IN THE
C     MANUAL MASTER FOR THE GIVEN PART NUMBER, AN ERROR
C     MESSAGE "NO RECORD FOR THAT PART EXISTS" IS PRINTED.
C     IF THERE IS AN ENTRY IN THE MANUAL MASTER BUT NO
C     ENTRIES IN THE CHAIN, THE USER IS GIVEN THE ERROR
C     MESSAGE "NO HISTORY EXISTS FOR THIS PART".  AFTER
C     THE PART HISTORY IS PRINTED, THE USER IS GIVEN THE
C     OPTION OF ENTERING ANOTHER PART NUMBER OR ENTERING
C     "EX".  IF THE USER CHOOSES "EX", HE WILL BE GIVEN
C     A CHANCE TO ENTER ANOTHER FUNCTION.
C
500   WRITE(1,510)
510   FORMAT("WHAT PART ARE YOU INTERESTED IN?"/
     1"(EX ENDS THIS FUNCTION)")
      READ(1,520)PART
520   FORMAT(4A2)
      IF(PART(1).EQ.2HEX)GO TO 595
      CALL DBFND(QA,FSET,MODE1,STAT,PARTN,PART)
      IF(STAT(1).EQ.0)GO TO 570
      IF(STAT(1).EQ.156)GO TO 550
      IF(STAT(1).NE.107)GO TO 535
      WRITE(1,530)
530   FORMAT("NO RECORD FOR THAT PART EXISTS")
      GO TO 500
535   WRITE(1,540)STAT(1)
540   FORMAT("PROGRAM ERROR ON FIND = ",I5)
      GO TO 1600
550   WRITE(1,560)
560   FORMAT("NO HISTORY EXISTS FOR THIS PART.")
      GO TO 500
570   WRITE(1,572)(PART(I),I=1,4)
572   FORMAT("PART FAILURE HISTORY FOR PART NUMBER ",4A2,/
     1"    ASSY#      DATE    PART#    CODE REF",/)
575   CALL DBGET(QA,FSET,MODE5,STAT,LALL,BUF,ARG)
      IF(STAT(1).EQ.0)GO TO 580
      IF(STAT(1).EQ.155)GO TO 500
      WRITE(1,579)STAT(1)
579   FORMAT("PROGRAM ERROR ON GET =",I5)
      GO TO 1600
580   WRITE(1,590)(BUF(I),I=1,16)
590   FORMAT(1X,5A2,1X,3A2,1X,4A2,1X,2A2,1X,2A2)
      GO TO 575
595   CONTINUE
      CALL DBCLS(QA,FSET,MODE2,STAT)
      IF(STAT(1).EQ.0)GO TO 345
      WRITE(1,597)STAT(1)
597   FORMAT("ERROR ON DATA SET CLOSE = ",I5)
      GO TO 1600
C
C     IF THE USER IS INTERESTED IN THE FAILURE COUNT FOR
C     A PARTICULAR ASSEMBLY, THIS SECTION IS ENTERED.
C     THE USER IS PROMPTED FOR AN ASSEMBLY AND THE
C     ASSEMBLY NUMBER AND THE NUMBER OF FAILURES IS
C     PRINTED.  THE USER IS PROMPTED FOR ASSEMBLY NUMBERS
C     UNTIL HE TYPES "EX".  AN "EX" RETURNS THE USER TO
C     A PROMPT FOR ANOTHER FUNCTION CHOICE.
C
600   WRITE(1,610)
610   FORMAT("IN WHICH ASSEMBLY ARE YOU INTERESTED?"/
     1"(EX ENDS THIS FUNCTION)")
      READ(1,620)ASMB
620   FORMAT(5A2)
      IF(ASMB(1).EQ.2HEX)GO TO 690
      CALL DBGET(QA,ASET,MODE7,STAT,LALL,BUF,ASMB)
      IF(STAT(1).EQ.0)GO TO 650
      IF(STAT(1).NE.107)GO TO 635
      WRITE(1,630)
630   FORMAT("NO RECORD EXISTS FOR THIS ASSEMBLY")
      GO TO 600
```

```
635    WRITE(1,640)STAT(1)
640    FORMAT("PROGRAM ERROR ON GET = ",I5)
       GO TO 1600
650    WRITE(1,660)(BUF(I),I=1,6)
660    FORMAT( "ASSY# = ",5A2,"  FAILURES = ",I5)
       GO TO 600
690    CONTINUE
       CALL DBCLS(QA,ASET,MODE2,STAT)
       IF(STAT(1).EQ.0)GO TO 345
       WRITE(1,695)
695    FORMAT("ERROR ON DATA SET CLOSE = ",I5)
       GO TO 1600
C
C      THIS SECTION READS THE MANUAL MASTER SERIALLY,
C      PICKING UP THE FAILURE COUNTS FOR EACH ASSEMBLY
C      NUMBER IN THE MASTER.  THE USER DOES NOT HAVE TO
C      ENTER ANYTHING.  AT THE END OF THE DISPLAY,
C      CONTROL IS RETURNED TO THE CHOICE OF FUNCTIONS.
C
700    CONTINUE
       ARG = 0
       CALL DBGET(QA,ASET,MODE4,STAT,LALL,BUF,ARG)
       WRITE(1,705)
705    FORMAT("ALL ASSEMBLIES FAILURE COUNTS"//
      1"     ASSY#     FAILS"/)
710    CALL DBGET(QA,ASET,MODE2,STAT,LALL,BUF,ARG)
       IF(STAT(1).EQ.0)GO TO 740
       IF(STAT(1).EQ.12)GO TO 780
       WRITE(1,720)STAT(1)
720    FORMAT("ERROR ON GET = ",I5)
       GO TO 1600
740    WRITE(1,750)(BUF(I),I=1,6)
750    FORMAT(1X,5A2,2X,I5)
       GO TO 710
780    CONTINUE
       CALL DBCLS(QA,ASET,MODE2,STAT)
       IF(STAT(1).EQ.0)GO TO 345
       WRITE(1,790)STAT(1)
790    FORMAT("ERROR ON DATA SET CLOSE = ",I5)
       GO TO 1600
800    WRITE(1,850)
850    FORMAT("YOU'VE HAD THREE TRIES AND COULDN'T ENTER A "/
      1"CORRECT FUNCTION.  SORRY - SEE YOU LATER!")
1600   CONTINUE
       CALL DBCLS(QA,ID,MODE1,STAT)
       IF(STAT(1).EQ.0)GO TO 1800
       WRITE(1,1700)STAT(1)
1700   FORMAT("ERROR ON CLOSE =",I5)
1800   CONTINUE
       END
       END$
```

```
FTN4,L
      PROGRAM PROJ
      INTEGER QA(7),LEVEL(3),STAT(10)
      INTEGER PROJ(3),DPROJ(3),ZERO,DATE(3)
      INTEGER DEPTN(3),ODATE(3),CDATE(3),PROJN(3)
      INTEGER P1,P2,D1,D2,OD1,OD2,CD1,CD2
      INTEGER ARRY(10,6),FLAG,RW,FUNC
      INTEGER BUF(11)
      DATA QA/2H  ,2HQA,2HDB,2H:1,2H00,2H:4,2H3;/
      DATA PROJ/2HPR,2HOJ,2H  /
      DATA PROJN/2HPR,2HOJ,2H# /
      DATA DEPTN/2HDE,2HPT,2H# /
      DATA ODATE/2HOP,2HDA,2HT /
      DATA CDATE/2HCL,2HDA,2HT /
      DATA LIST/2H@ /
      DATA ZERO/0/
      DATA RW/2HRW/
      DATA MODE1/1/
      DATA MODE2/2/
      DATA MODE7/7/
C
C     THIS PROGRAM UPDATES THE PROJECT FILE.  THE USER CAN
C     ENTER A PROJECT NUMBER, DELETE A PROJECT NUMBER
C     OR CLOSE A PROJECT OUT BY ENTERING A CLOSE DATE.
C     ADDING A PROJECT NUMBER CONSISTS OF ADDING AN ENTRY
C     TO THE PROJ SET.  DELETING A PROJECT NUMBER CONSISTS
C     OF DELETING THE ENTRY FROM THE SET.  IF THERE ARE
C     DETAIL RECORDS LINKED TO THIS PROJECT NUMBER, THE
C     NUMBER CANNOT BE DELETED.  IF THE PROJECT IS CLOSED,
C     THE RECORD IS MODIFIED TO INCLUDE A CLOSE DATE.
C     THE USER IS ASKED FOR A LEVEL WORD AND AN ATTEMPT
C     IS MADE TO OPEN THE DATABASE AND THE USER IS
C     INFORMED IF THE LEVEL GIVEN IS INADEQUATE TO
C     ACCESS ANY ITEMS IN THE DATABASE.
C
      WRITE(1,100)
100   FORMAT("QA PROJECT NUMBER PROGRAM"/
     1"ENTER YOUR LEVEL WORD.")
      READ(1,105)(LEVEL(I),I=1,3)
105   FORMAT(3A2)
      CALL DBOPN(QA,LEVEL,MODE1,STAT)
      IF(STAT(1).EQ.0)GO TO 335
      IF(STAT(1).EQ.153)GO TO 330
      WRITE(1,325)STAT(1)
325   FORMAT("SYSTEM ERROR = ",I5," ON OPEN."/
     1"CALL A SYSTEMS ANALYST.")
      GO TO 2000
330   CONTINUE
      WRITE(1,331)
331   FORMAT("YOUR LEVEL DOESN'T ALLOW SUFFICIENT ACCESS FOR ANY
     1TRANSACTION.")
```

```
      GO TO 2000
C
C     BEFORE THE USER CAN ENTER ANY INFORMATION, THE
C     SUBROUTINE INFO IS CALLED TO BUILD A TABLE OF
C     LENGTHS AND OFFSETS OF EACH ITEM IN THE DATA SET.
C     IF THE NUMBER OF ITEMS IN THE DATA SET IS
C     MORE THAN THE TABLE SPACE ALLOWED FOR IN
C     INFO, THE USER IS SO INFORMED.
C
335   CALL INFO(QA,7,PROJ,ARRY,FLAG)
      IF(FLAG.EQ.0)GO TO 345
      IF(FLAG.EQ.1)GO TO 337
      WRITE(1,336)FLAG
336   FORMAT("SYSTEM ERROR = ",I5," ON DB INFO CALL."/
     1"CALL A SYSTEMS ANALYST.")
      GO TO 1600
337   WRITE(1,338)
338   FORMAT("TOO MANY ITEMS IN DATA SET FOR TABLE."/
     1"CALL A SYSTEMS ANALYST.")
      GO TO 1600
C
C     THE USER IS ASKED TO SELECT A FUNCTION AND IS
C     GIVEN THREE CHANCES TO ENTER A CORRECT FUNCTION.
C     ONCE THE FUNCTION IS ENTERED, THE PROGRAM
C     BRANCHES TO THE PROPER SECTION FOR PROCESSING.
C
345   ICNT = 0
346   WRITE(1,347)
347   FORMAT("CHOOSE A FUNCTION."/
     1"AD = ADD A PROJECT"/"DE = DELETE A PROJECT"/
     2"CL = CLOSE A PROJECT"/"EP = END PROGRAM")
      READ(1,348)FUNC
348   FORMAT(A2)
      IF(FUNC.EQ.2HAD)GO TO 500
      IF(FUNC.EQ.2HDE)GO TO 600
      IF(FUNC.EQ.2HCL)GO TO 700
      IF(FUNC.EQ.2HEP)GO TO 1600
350   ICNT=ICNT+1
      IF(ICNT.EQ.3)GO TO 1700
      WRITE(1,400)
400   FORMAT("FUNCTION INPUT ERROR - TRY AGAIN!")
      GO TO 346
C
C     THIS SECTION PROCESSES THE ADDING OF A
C     PROJECT NUMBER.  FIRST THE SUBROUTINE OFSET
C     IS CALLED TO DETERMINE THE STARTING AND
C     ENDING CHARACTER POSITIONS OF EACH ITEM
C     AND IF THE USER HAS READ ONLY OR READ/
C     WRITE ACCESS.  IF THE USER DOES NOT HAVE
C     WRITE ACCESS TO EACH ITEM, AN ENTRY CANNOT
C     BE ADDED.  IF THE USER CAN ADD AN ENTRY
```

```
C     THE PROGRAM PROMPTS FOR EACH ITEM IN THE
C     ENTRY.  IF THE ENTRY CANNOT BE ADDED
C     BECAUSE THE KEY ALREADY EXISTS, OR BECAUSE
C     THE SET IS FILLED TO CAPACITY, THE USER
C     IS SO INFORMED AND THE PROGRAM GOES BACK
C     TO ASK FOR ANOTHER ENTRY TO ADD.  ANY
C     OTHER ERRORS CAUSE THE PROGRAM TO BE
C     TERMINATED.
C
500   CALL OFSET(ARRY,PROJN,P1,P2,RW,FLAG)
      IF(FLAG.NE.0)GO TO 790
      CALL OFSET(ARRY,DEPTN,D1,D2,RW,FLAG)
      IF(FLAG.NE.0)GO TO 790
      CALL OFSET(ARRY,ODATE,OD1,OD2,RW,FLAG)
      IF(FLAG.NE.0)GO TO 790
      CALL OFSET(ARRY,CDATE,CD1,CD2,RW,FLAG)
      IF(FLAG.NE.0)GO TO 790
      WRITE(1,505)
505   FORMAT("WHAT PROJECT NUMBER DO YOU WANT TO ADD?"/
     1"(EX ENDS THIS FUNCTION.)")
      READ(1,510)(BUF(I),I=P1,P2)
510   FORMAT(3A2)
      IF(BUF(P1).EQ.2HEX)GO TO 560
      WRITE(1,515)
515   FORMAT("WHAT IS THE DEPARTMENT NUMBER?")
      READ(1,520)(BUF(I),I=D1,D2)
520   FORMAT(2A2)
      WRITE(1,525)
525   FORMAT("WHAT IS THE OPEN DATE?")
      READ(1,530)(BUF(I),I=OD1,OD2)
530   FORMAT(3A2)
      DO 535 I=CD1,CD2
      BUF(I)=2H
535   CONTINUE
      CALL DBLCK(QA,ID,MODE1,STAT)
      IF(STAT(1).EQ.0)GO TO 545
      WRITE(1,540)STAT(1)
540   FORMAT("SYSTEM ERROR = ",I5," ON LOCK."/
     1"CALL A SYSTEMS ANALYST.")
      GO TO 1600
545   CONTINUE
      CALL DBPUT(QA,PROJ,MODE1,STAT,LIST,BUF)
      IF(STAT(1).EQ.0)GO TO 555
      IF(STAT(1).EQ.106.OR.STAT(1).EQ.110)GO TO 553
      WRITE(1,550)STAT(1)
550   FORMAT("SYSTEM ERROR = ",I5," ON PUT."/
     1"CALL A SYSTEMS ANALYST.")
      GO TO 1600
553   WRITE(1,554)
554   FORMAT("DATABASE IS FULL OR PROJECT NUMBER ALREADY EXISTS.")
555   CALL DBUNL(QA,ID,MODE1,STAT)
```

```
      IF(STAT(1).EQ.0)GO TO 500
556   WRITE(1,557)STAT(1)
557   FORMAT("SYSTEM ERROR = ",I5," ON UNLOCK."/
     1"CALL A SYSTEMS ANALYST.")
      GO TO 500
560   CALL DBCLS(QA,PROJ,MODE2,STAT)
      IF(STAT(1).EQ.0)GO TO 345
      WRITE(1,565)STAT(1)
565   FORMAT("SYSTEM ERROR = ",I5 " ON DATA SET CLOSE."/
     1"CALL A SYSTEMS ANALYST.")
      GO TO 1600
C
C     IF THE USER SELECTS "DE" AS A FUNCTION, THE
C     PROGRAM BRANCHES TO THIS LOCATION.  HERE THE
C     USER IS PROMPTED FOR THE PROJECT NUMBER TO
C     DELETE.  IF THE ENTRY HAS DETAIL ENTRIES
C     CHAINED TO THIS MASTER ENTRY, THE USER IS
C     INFORMED AND THE PROGRAM PROMPTS FOR
C     ANOTHER NUMBER TO BE DELETED.  ANY OTHER
C     ERROR CAUSES THE PROGRAM TO BE TERMINATED.
C
600   WRITE(1,605)
605   FORMAT("WHAT IS THE PROJECT NUMBER TO BE DELETED?"/
     1"(EX ENDS THIS FUNCTION.)")
      READ(1,610)(DPROJ(I),I=1,3)
610   FORMAT(3A2)
      IF(DPROJ(1).EQ.2HEX)GO TO 660
      CALL DBLCK(QA,ID,MODE1,STAT)
      IF(STAT(1).EQ.0)GO TO 620
613   WRITE(1,615)
615   FORMAT("SYSTEM ERROR = ",I5," ON LOCK."/
     1"CALL A SYSTEMS ANALYST.")
      GO TO 1600
620   CALL DBGET(QA,PROJ,MODE7,STAT,ZERO,BUF,DPROJ)
      IF(STAT(1).EQ.0)GO TO 635
      IF(STAT(1).NE.107)GO TO 625
      WRITE(1,621)
621   FORMAT("THIS PROJECT NUMBER DOES NOT EXIST.")
622   CALL DBUNL(QA,ID,MODE1,STAT)
      IF(STAT(1).EQ.0)GO TO 600
      GO TO 556
625   WRITE(1,626)STAT(1)
626   FORMAT("SYSTEM ERROR = ",I5," ON GET."/
     1"CALL A SYSTEMS ANALYST.")
      GO TO 1600
635   CALL DBDEL(QA,PROJ,MODE1,STAT)
      IF(STAT(1).EQ.0)GO TO 622
      IF(STAT(1).EQ.113)GO TO 642
      WRITE(1,640)STAT(1)
640   FORMAT("SYSTEM ERROR = ",I5," ON DELETE."/
     1"CALL A SYSTEMS ANALYST.")
```

```
      GO TO 1600
642   WRITE(1,645)
645   FORMAT("TRANSACTIONS STILL EXIST FOR THIS PROJECT NUMBER."/
     1"IT CANNOT BE DELETED.")
      GO TO 622
660   CALL DBCLS(QA,PROJ,MODE2,STAT)
      IF(STAT(1).EQ.0)GO TO 345
      WRITE(1,665)STAT(1)
665   FORMAT("SYSTEM ERROR = ",I5," ON DATA SET CLOSE."/
     1"CALL A SYSTEMS ANALYST.")
      GO TO 1600
C
C     IF THE USER WANTS TO CLOSE A PROJECT NUMBER, THE
C     PROGRAM BRANCHES HERE.  THE USER IS PROMPTED FOR
C     THE PROJECT NUMBER TO BE CLOSED AND THE PROGRAM
C     ATTEMPTS TO ACCESS THE ENTRY.  IF THE ENTRY DOES
C     NOT EXIST, THE USER IS SO INFORMED AND IS ASKED
C     FOR ANOTHER PROJECT TO BE CLOSED.  OTHERWISE,
C     THE CLOSING DATE IS ADDED TO THE ENTRY AND THE
C     ENTRY UPDATED.
C
700   CONTINUE
      CALL OFSET(ARRY,PROJN,P1,P2,RW,FLAG)
      IF(FLAG.NE.0)GO TO 790
      CALL OFSET(ARRY,CDATE,CD1,CD2,RW,FLAG)
      IF(FLAG.NE.0)GO TO 790
      WRITE(1,705)
705   FORMAT("WHAT PROJECT NUMBER DO YOU WANT CLOSED?"/
     1"(EX ENDS THIS FUNCTION.)")
      READ(1,710)(DPROJ(I),I=1,3)
710   FORMAT(3A2)
      IF(DPROJ(1).EQ.2HEX)GO TO 660
      WRITE(1,715)
715   FORMAT("WHAT IS THE CLOSING DATE?")
      READ(1,720)(DATE(I),I=1,3)
720   FORMAT(3A2)
      CALL DBLCK(QA,ID,MODE1,STAT)
      IF(STAT(1).NE.0)GO TO 613
      CALL DBGET(QA,PROJ,MODE7,STAT,LIST,BUF,DPROJ)
      IF(STAT(1).EQ.0)GO TO 730
      IF(STAT(1).NE.107)GO TO 625
      WRITE(1,725)
725   FORMAT("THIS PROJECT NUMBER DOESN'T EXIST.")
      CALL DBUNL(QA,ID,MODE1,STAT)
      IF(STAT(1).EQ.0)GO TO 700
      GO TO 556
730   CONTINUE
      J=P1
      K=CD1
      DO 735 I=1,3
      BUF(J)=DPROJ(I)
      BUF(K)=DATE(I)
      J=J+1
      K=K+1
735   CONTINUE
      CALL DBUPD(QA,PROJ,MODE1,STAT,LIST,BUF)
      IF(STAT(1).EQ.0)GO TO 700
      WRITE(1,740)STAT(1)
740   FORMAT("SYSTEM ERROR = ",I5," ON DB UPDATE."/
     1"CALL A SYSTEMS ANALYST.")
      GO TO 1600
790   WRITE(1,795)
795   FORMAT("YOUR LEVEL DOESN'T ALLOW SUFFICIENT ACCESS FOR THIS
     1TRANSACTION.")
      GO TO 345
1600  CALL DBCLS(QA,ID,MODE1,STAT)
      IF(STAT(1).EQ.0)GO TO 2000
      WRITE(1,1650)STAT(1)
1650  FORMAT("SYSTEM ERROR = ",I5," ON DATABASE CLOSE."/
     1"CALL A SYSTEMS ANALYST.")
      GO TO 2000
1700  WRITE(1,1750)
1750  FORMAT("YOU'VE HAD THREE TRIES AND COULDN'T ENTER A "/
     1"CORRECT FUNCTION.  SORRY - SEE YOU LATER!")
2000  CONTINUE
      END
      END$
```

```
FTN4,L
      PROGRAM TRANS
      INTEGER LEVEL(3),QA(7),STAT(10)
      INTEGER BUF(23),BUF2(6)
      INTEGER FAIL(3),ASMBY(3),ASSYN(5)
      INTEGER ASSY(3),PART(3),FAILCD(3),DATE(3),REFDES(3),PNTS(3)
      INTEGER EMP(3),FAILN(3)
      INTEGER ARRY(10,6),ARRY2(10,6),FLAG,RW
      INTEGER A1,A2,D1,D2,P1,P2,F1,F2,R1,R2,PT1,PT2,E1,E2,FN1,FN2
      DATA QA/2H  ,2HQA,2HDB,2H:1,2H00,2H:4,2H3;/
      DATA FAIL/2HFA,2HIL,2H  /
      DATA ASMBY/2HAS,2HMB,2HY /
      DATA ASSY/2HAS,2HSY,2H# /
      DATA PART/2HPA,2HRT,2H# /
      DATA RW/2HRW/
      DATA FAILCD/2HFA,2HIL,2HCD/
      DATA REFDES/2HRE,2HFD,2HES/
      DATA PNTS/2HPN,2HTS,2H  /
      DATA EMP/2HEM,2HP#,2H  /
      DATA FAILN/2HFA,2HIL,2H# /
      DATA DATE/2HDA,2HTE,2H  /
      DATA LIST/2H@ /
      DATA ICODE/100/
      DATA MODE1/1/
      DATA MODE7/7/
C
C     THIS PROGRAM ALLOWS AN OPERATOR TO ENTER A FAILURE
C     INTO THE FAILURE DATA SET.  EACH ADDITION TO THE
C     FAILURE DATA SET ALSO CAUSES THE MANUAL MASTER
C     FAILURE COUNT ITEM TO BE UPDATED.  BEFORE THE USER
C     CAN ENTER A TRANSACTION, HE MUST ENTER A SECURITY
C     CODE AND LEVEL WORD.  THE SECURITY CODE MUST MATCH
C     THE SECURITY CODE IN THE PROGRAM.  IF THE LEVEL
C     CODE WORD SUBMITTED DOES NOT ALLOW ACCESS TO ANY
C     ITEMS IN THE DATABASE, THE USER IS INFORMED AND
C     THE PROGRAM TERMINATES.
C
      WRITE(1,100)
100   FORMAT("PLEASE ENTER PROPER SECURITY CODE.")
      READ(1,*)ISEC
      IF(ISEC.EQ.ICODE)GO TO 400
      WRITE(1,300)
300   FORMAT("BAD SECURITY CODE - GOOD-BYE!")
      GO TO 9900
400   WRITE(1,500)
500   FORMAT("PLEASE ENTER YOUR LEVEL.")
      READ(1,600)(LEVEL(I),I=1,3)
600   FORMAT(3A2)
      CALL DBOPN(QA,LEVEL,MODE1,STAT)
      IF(STAT(1).EQ.0)GO TO 710
      IF(STAT(1).NE.153)GO TO 750
      WRITE(1,700)
700   FORMAT("YOUR LEVEL DOESN'T ALLOW ACCESS TO THE DATABASE."/
     1"  GOOD-BYE!")
      GO TO 9900
C
C     THE SUBROUTINE INFO IS CALLED TO SET UP THE TABLE
C     OF ITEM LENGTHS AND OFSETS.  THE TABLE WILL ALSO
C     SPECIFY IF THE USER HAS READ OR READ/WRITE ACCESS
C     TO THE ITEM.  THE TABLE IN THE INFO ROUTINE IS A
C     FIXED LENGTH AND IF FOR SOME REASON, THE NUMBER
C     OF ITEMS IN THE ENTRY IS GREATER THAN THE TABLE
C     SIZE, INFO RETURNS AN ERROR CODE AND THIS PROGRAM
C     REPORTS IT.  INFO MUST BE CALLED FOR BOTH OF THE
C     SETS THAT ARE TO BE ACCESSED.  ONCE THE TABLES
C     ARE SET UP FOR INFO, THE SUBROUTINE OFSET MUST
C     BE CALLED FOR EACH ITEM IN A FAILURE TRANSACTION.
C     IF THE USER DOES NOT HAVE WRITE ACCESS FOR EACH
C     ITEM, A TRANSACTION CANNOT BE ENTERED AND THIS
C     ERROR IS REPORTED.
C
710   CALL INFO(QA,7,FAIL,ARRY,FLAG)
      IF(FLAG.EQ.0)GO TO 730
      IF(FLAG.GT.0)GO TO 725
      FLAG=IABS(FLAG)
      WRITE(1,720)FLAG
720   FORMAT("SYSTEM ERROR = ",I5," ON INF. CALL."/
     1"  CALL A SYSTEMS ANALYST.")
      GO TO 1600
725   WRITE(1,728)
728   FORMAT("INFO TABLE IS TOO SMALL."/
     1"CALL A SYSTEMS ANALYST.")
      GO TO 1600
730   CALL INFO(QA,7,ASMBY,ARRY2,FLAG)
      IF(FLAG.EQ.0)GO TO 735
      IF(FLAG.GT.0)GO TO 725
      WRITE(1,720)FLAG
      GO TO 1600
735   CALL OFSET(ARRY,ASSY,A1,A2,RW,FLAG)
      IF(FLAG.NE.0)GO TO 740
      CALL OFSET(ARRY,DATE,D1,D2,RW,FLAG)
      IF(FLAG.NE.0)GO TO 740
      CALL OFSET(ARRY,PART,P1,P2,RW,FLAG)
      IF(FLAG.NE.0)GO TO 740
      CALL OFSET(ARRY,FAILCD,F1,F2,RW,FLAG)
      IF(FLAG.NE.0)GO TO 740
      CALL OFSET(ARRY,REFDES,R1,R2,RW,FLAG)
      IF(FLAG.NE.0)GO TO 740
      CALL OFSET(ARRY,PNTS,PT1,PT2,RW,FLAG)
      IF(FLAG.NE.0)GO TO 740
      CALL OFSET(ARRY,EMP,E1,E2,RW,FLAG)
      IF(FLAG.NE.0)GO TO 740
      CALL OFSET(ARRY2,FAILN,FN1,FN2,RW,FLAG)
      IF(FLAG.NE.0)GO TO 740
      GO TO 800
740   WRITE(1,745)
745   FORMAT("YOUR LEVEL DOESN'T ALLOW YOU TO ENTER A TRANSACTION."
     1" GOOD-BYE!")
      GO TO 1600
750   WRITE(1,770)STAT(1)
770   FORMAT("SYSTEM ERROR = ",I5," ON OPEN."/
     1"CALL A SYSTEMS ANALYST.")
      GO TO 9900
C
C     THE USER IS THEN PROMPTED FOR EACH ENTRY IN
C     THE TRANSACTION.  THE VALUES THAT ARE ENTERED
C     ARE PLACED IN THE BUFFER BASED ON THE OFSET
C     VALUES RETURNED FROM THE OFSET SUBROUTINE.
C     ONCE THE BUFFER IS FULL, THE MANUAL MASTER
C     RECORD IS ACCESSED TO UPDATE THE FAILURE
C     COUNT, AND THE FAILURE TRANSACTION IS ADDED
C     TO THE FAILURE SET.
C
800   WRITE(1,900)
900   FORMAT("GET READY TO ENTER A TRANSACTION."/
     1"EN ENDS THIS PROGRAM."/
     2"ENTER ASSEMBLY NUMBER.")
      READ(1,1000)(BUF(I),I=A1,A2)
1000  FORMAT(5A2)
      J = 1
      DO 1005 I=A1,A2
      ASSYN(J) = BUF(I)
      J = J + 1
1005  CONTINUE
      IF(BUF(A1).EQ.2HEN)GO TO 1600
      WRITE(1,1010)
1010  FORMAT("ENTER DATE AS YYMMDD")
      READ(1,1020)(BUF(I),I=D1,D2)
1020  FORMAT(3A2)
      WRITE(1,1030)
1030  FORMAT("ENTER PART NUMBER")
      READ(1,1040)(BUF(I),I=P1,P2)
1040  FORMAT(4A2)
      WRITE(1,1050)
1050  FORMAT("ENTER FAILED CODE")
      READ(1,1060)(BUF(I),I=F1,F2)
1060  FORMAT(2A2)
      WRITE(1,1070)
1070  FORMAT("ENTER REFERENCE DESIGNATOR")
      READ(1,1080)(BUF(I),I=R1,R2)
1080  FORMAT(2A2)
      WRITE(1,1090)
1090  FORMAT("ENTER POINT VALUES (0 OR 1)")
```

```
      READ(1,1100)(BUF(I),I=PT1,PT2)
1100  FORMAT(4A2)
      WRITE(1,1110)
1110  FORMAT("ENTER EMPLOYEE NUMBER")
      READ(1,1120)(BUF(I),I=E1,E2)
1120  FORMAT(3A2)
      CALL DBLCK(QA,ID,MODE1,STAT)
      IF(STAT(1).NE.0)GO TO 1500
      CALL DBGET(QA,ASMBY,MODE7,STAT,LIST,BUF2,ASSYN)
      IF(STAT(1).NE.0)GO TO 1510
      CALL DBPUT(QA,FAIL,MODE1,STAT,LIST,BUF)
      IF(STAT(1).NE.0)GO TO 1520
      BUF2(FN1) = BUF2(FN1) + 1
      CALL DBUPD(QA,ASMBY,MODE1,STAT,LIST,BUF2)
      IF(STAT(1).NE.0)GO TO 1530
1200  CALL DBUNL(QA,ID,MODE1,STAT)
      IF(STAT(1).EQ.0)GO TO 800
C
C     IF THE PART NUMBER OR THE ASSEMBLY NUMBER ENTERED WAS
C     ILLEGAL, THIS IS REPORTED AND THE USER CAN ENTER
C     ANOTHER TRANSACTION.  ANY OTHER ERROR IS A SYSTEM
C     ERROR AND THE PROGRAM IS TERMINATED.  IF THERE WAS
C     A SYSTEM ERROR ON UPDATING THE MANUAL MASTER, THE
C     USER IS NOTIFIED OF THIS FACT AND OF THE FACT THAT
C     THE ENTRY IN THE DETAIL DATA SET WAS ALREADY ENTERED.
C     THIS TELLS HIM THAT THE MASTER AND THE DETAIL DO
C     NOT AGREE AS FAR AS COUNT.
C
      WRITE(1,1300)STAT(1)
1300  FORMAT("SYSTEM ERROR = ",I5,"ON UNLOCK."/
     1"CALL A SYSTEMS ANALYST.")
      GO TO 1600
1500  WRITE(1,1505)STAT(1)
1505  FORMAT("SYSTEM ERROR = ",I5," ON LOCK."/
     1"CALL A SYSTEMS ANALYST.")
      GO TO 1600
1510  IF(STAT(1).EQ.107)GO TO 1517
      WRITE(1,1515)STAT(1)
1515  FORMAT("SYSTEM ERROR = ",I5," ON GET."/
     1"CALL A SYSTEMS ANALYST.")
      GO TO 1600
1517  WRITE(1,1518)
1518  FORMAT("ILLEGAL ASSEMBLY NUMBER.")
      GO TO 1200
1520  IF(STAT(1).EQ.107)GO TO 1527
      WRITE(1,1525)STAT(1)
1525  FORMAT("SYSTEM ERROR = ",I5," ON PUT."/
     1"CALL A SYSTEMS ANALYST.")
      GO TO 1600
1527  WRITE(1,1528)
1528  FORMAT("ILLEGAL PART NUMBER.")
```

```
      GO TO 1200
1530  WRITE(1,1535)STAT(1)
1535  FORMAT("SYSTEM ERROR = ",I5," ON UPDATE."/
     1"HISTORY RECORD WAS ALREADY ENTERED."/
     2"CALL A SYSTEMS ANALYST.")
1600  CALL DBCLS(QA,ID,MODE1,STAT)
      IF(STAT.EQ.0)GO TO 9900
      WRITE(1,1610)STAT(1)
1610  FORMAT("SYSTEM ERROR = ",I5," ON CLOSE."/
     1"CALL A SYSTEMS ANALYST.")
9900  CONTINUE
      END
      END$
```

```
FTN4,L
      SUBROUTINE INFO(DBASE,BASDM,DSET,ARRAY,FLAG)
      INTEGER DBASE(BASDM)
      INTEGER DSET(3)
      INTEGER BASDM
      INTEGER BUF(256)
      INTEGER STAT(10)
      INTEGER BUF2(13)
      INTEGER FLAG
      INTEGER ARRAY(10,6)
      INTEGER CUM
      DO 100 I=1,10
      DO 100 J=1,6
      ARRAY(I,J)=2H
100   CONTINUE
      CALL DBINF(DBASE,DSET,104,STAT,BUF)
      IF(STAT(1).NE.0)GO TO 300
      N=BUF(1)+1
      IF(N.GT.10)GO TO 400
      CUM=1
      DO 200 I=2,N
      M=I-1
      CALL DBINF(DBASE,IABS(BUF(I)),102,STAT,BUF2)
      IF(STAT(1).NE.0)GO TO 300
      ARRAY(M,1)=BUF2(1)
      ARRAY(M,2)=BUF2(2)
      ARRAY(M,3)=BUF2(3)
      ARRAY(M,4)=BUF2(10)*BUF2(11)
      IF(BUF2(9).EQ.2HX )ARRAY(M,4)=ARRAY(M,4)/2
      ARRAY(M,5)=CUM
      IF(BUF(I).LT.0)ARRAY(M,6)=2H$$
      CUM=CUM+ARRAY(M,4)
200   CONTINUE
      FLAG=0
      GO TO 900
C
C          IF THERE WAS AN ERROR IN THE DBINF CALL, THE FLAG
C          VALUE RETURNED IS THE NEGATIVE OF THE IMAGE ERROR CODE
C
300   FLAG=-STAT(1)
      GO TO 900
C
C          IF THE TABLE IS NOT LARGE ENOUGH TO HANDLE ALL THE
C          ITEMS, A FLAG VALUE OF +1 IS RETURNED
C
400   FLAG=+1
900   CONTINUE
      RETURN
      END
      END$
```

```
FTN4,L
      SUBROUTINE OFSET(ARRAY,NAME,BEG,END,RW,FLAG)
      INTEGER ARRAY(10,6)
      INTEGER NAME(3)
      INTEGER FLAG,RW
      INTEGER BEG
      INTEGER END
C
C     THIS SUBROUTINE LOOKS FOR AN ITEM IN THE TABLE BUILT
C     BY INFO AND RETURNS THE BEGINNING AND ENDING
C     CHARACTER POSITIONS.  IF RW IS NOT "RO" (READ ONLY)
C     BUT THE USER DOES NOT HAVE WRITE ACCESS TO THE ITEM,
C     FLAG IS SET TO 1.  FLAG IS ALSO SET TO 1 IF THE
C     ITEM IS NOT IN THE TABLE.
C
      DO 300 I=1,10
      IF(ARRAY(I,1).EQ.2H  )GO TO 400
      DO 200 J=1,3
      IF(ARRAY(I,J).NE.NAME(J))GO TO 300
200   CONTINUE
      IF(RW.EQ.2HRO)GO TO 250
      IF(ARRAY(I,6).NE.2H$$)GO TO 300
250   END=ARRAY(I,4)+ARRAY(I,5)-1
      BEG=ARRAY(I,5)
      FLAG=0
      RETURN
300   CONTINUE
400   FLAG=1
      RETURN
      END
      END$
```

Pascal Examples

The following example shows external declarations in PASCAL for IMAGE subroutines. No attempt is made to show programming examples, as such examples would be similar to the previously given FORTRAN programs PART, PROJ, and TRANS.

If a PASCAL program uses both IMAGE and the heap/stack area, space must be reserved for IMAGE's dynamic storage (run table, record buffer, and FMP DCBs) area by using the compiler option $IMAGE n$, where n is the number of words to be reserved.

See Chapter 7 to calculate n for a specific database. In general, n = 2000 will allow all but the largest databases to be accessed, while those may require up to 10,240 words.

Failure to invoke the $IMAGE n$ compiler option, or not making n large enough, will result in a run-time error 128 from DBOPN: "Not enough space for IMAGE data buffers." This should not be confused with the run-time error "INSUFFICIENT IMAGE SPACE" from PASCAL, which means that the partition in which the program is executing is too small.

If the PASCAL program uses the $HEAP 0$ compiler option, then it is not necessary to include the $IMAGE n$ command explicitly. This may be advantageous if the size of the dynamic storage area cannot be determined at compile time.

```
$IMAGE 2000$

TYPE
        SINGLE_INTEGER  = -32768..32767;
        IBASE_TYPE      = PACKED ARRAY [1..16] OF CHAR;
        ASCII_NAME      = PACKED ARRAY [1..6] OF CHAR;
        STATUS_TYPE     = ARRAY [1..10] OF SINGLE_INTEGER;
        LIST_TYPE       = PACKED ARRAY [1..250] OF CHAR;
        BUFFER_TYPE     = PACKED ARRAY [1..500] OF CHAR;
        KEY_TYPE        = PACKED ARRAY [1..40] OF CHAR;

{External declaration of all IMAGE subroutines (procedures)}

PROCEDURE dbopn (VAR ibase : IBASE_TYPE;
                 VAR ilevl : ASCII_NAME;
                 VAR imode : SINGLE_INTEGER;
                 VAR istat : STATUS_TYPE);  EXTERNAL;
```

```
PROCEDURE dbinf (VAR ibase : IBASE_TYPE;
                 VAR id    : ASCII_NAME;
                 VAR imode : SINGLE_INTEGER;
                 VAR istat : STATUS_TYPE;
                 VAR ibuf  : BUFFER_TYPE);  EXTERNAL;

PROCEDURE dbfnd (VAR ibase : IBASE_TYPE;
                 VAR id    : ASCII_NAME;
                 VAR imode : SINGLE_INTEGER;
                 VAR istat : STATUS_TYPE;
                 VAR item  : ASCII_NAME;
                 VAR iarg  : KEY_TYPE);     EXTERNAL;

PROCEDURE dbget (VAR ibase : IBASE_TYPE;
                 VAR id    : ASCII_NAME;
                 VAR imode : SINGLE_INTEGER;
                 VAR istat : STATUS_TYPE;
                 VAR list  : LIST_TYPE;
                 VAR ibuf  : BUFFER_TYPE;
                 VAR iarg  : KEY_TYPE);     EXTERNAL;

PROCEDURE dbput (VAR ibase : IBASE_TYPE;
                 VAR id    : ASCII_NAME;
                 VAR imode : SINGLE_INTEGER;
                 VAR istat : STATUS_TYPE;
                 VAR list  : LIST_TYPE;
                 VAR ibuf  : BUFFER_TYPE);  EXTERNAL;

PROCEDURE dbupd (VAR ibase : IBASE_TYPE;
                 VAR id    : ASCII_NAME;
                 VAR imode : SINGLE_INTEGER;
                 VAR istat : STATUS_TYPE;
                 VAR list  : LIST_TYPE;
                 VAR ibuf  : BUFFER_TYPE);  EXTERNAL;

PROCEDURE dbdel (VAR ibase : IBASE_TYPE;
                 VAR id    : ASCII_NAME;
                 VAR imode : SINGLE_INTEGER;
                 VAR istat : STATUS_TYPE);  EXTERNAL;

PROCEDURE dblck (VAR ibase : IBASE_TYPE;
                 VAR id    : ASCII_NAME;
                 VAR imode : SINGLE_INTEGER;
                 VAR istat : STATUS_TYPE);  EXTERNAL;

PROCEDURE dbunl (VAR ibase : IBASE_TYPE;
                 VAR id    : ASCII_NAME;
                 VAR imode : SINGLE_INTEGER;
                 VAR istat : STATUS_TYPE);  EXTERNAL;
```

```
PROCEDURE dbcls (VAR ibase : IBASE_TYPE;
                 VAR id    : ASCII_NAME;
                 VAR imode : SINGLE_INTEGER;
                 VAR istat : STATUS_TYPE);  EXTERNAL;
```

Assembly Language

In the following sections, the calling sequences from Assembly Language for the various subroutines are shown. No attempt is made to show programming examples, as such examples would be extremely lengthy and application specific. For more information about any parameters shown, see Chapter 4.

Open Database

```
JSB DBOPN
DEF *+5
DEF IBASE
DEF ILEVL
DEF IMODE
DEF ISTAT
```

Request Database Information

```
JSB DBINF
DEF *+6
DEF IBASE
DEF ID
DEF IMODE
DEF ISTAT
DEF IBUF
```

Locate An Entry

```
JSB DBFND
DEF *+7
DEF IBASE
DEF ID
DEF IMODE
DEF ISTAT
DEF ITEM
DEF IARG
```

Read An Entry

```
JSB DBGET
DEF *+8
DEF IBASE
DEF ID
DEF IMODE
DEF ISTAT
DEF LIST
DEF IBUF
DEF IARG
```

Add An Entry

```
JSB DBPUT
DEF *+7
DEF IBASE
DEF ID
DEF IMODE
DEF ISTAT
DEF LIST
DEF IBUF
```

Update An Entry

```
JSB DBUPD
DEF *+7
DEF IBASE
DEF ID
DEF IMODE
DEF ISTAT
DEF LIST
DEF IBUF
```

Delete An Entry

```
JSB DBDEL
DEF *+5
DEF IBASE
DEF ID
DEF IMODE
DEF ISTAT
```

Lock And Unlock Database

```
        JSB DBLCK
        DEF *+5
        DEF IBASE
        DEF ID
        DEF IMODE
        DEF ISTAT

        JSB DBUNL
        DEF *+5
        DEF IBASE
        DEF ID
        DEF IMODE
        DEF ISTAT
```

Close A Database

```
        JSB DBCLS
        DEF *+5
        DEF IBASE
        DEF ID
        DEF IMODE
        DEF ISTAT
```

BASIC Examples

```
+------------------------------+
|             NOTE             |
|                              |
|  These  examples  are  not   |
|  applicable to 92073A IMAGE  |
+------------------------------+
```

The examples below are either designed specifically to illustrate the procedure discussed or are taken from the program TRANS. TRANS is shown in its entirety at the end of this section.

Open Database

```
10 DIM S[10]
30 DIM Q$[14],S$[6],F$[6],L$[6]
60 DATA 1,7,100,201
70 DATA "  QA:100:43;"
120 READ M1,M7,I1,M2
130 READ Q$
180 PRINT "PLEASE ENTER PROPER SECURITY CODE."
190 INPUT S1
200 IF S1#I1 THEN 880
210 PRINT "PLEASE ENTER YOUR LEVEL."
220 INPUT L$
230 CALL DBOPN(Q$,L$,M1,S[1])
240 IF S[1]#0 THEN 810
          .
          .
          .
810 IF S[1]#153 THEN 850
820 PRINT "YOUR LEVEL DOESN'T ALLOW YOU TO ENTER A TRANSACTION."
840 GOTO 890
850 PRINT "SYSTEM ERROR = ",S[1]," ON OPEN."
860 PRINT "CALL A SYSTEMS ANALYST."
870 GOTO 890
880 PRINT "BAD SECURITY CODE - GOODBYE."
890 END
```

In this example, the QA database is opened after the user enters the security code and level. Later, a subroutine will be used to determine if the user's level is sufficient to enter a transaction.

The database is opened in mode 1, which allows read/write access concurrent with other programs opened in mode 1. It could also be possible to open in mode 3, which would request exclusive read/write access.

The first two characters of the array containing the database name is blank, since BASIC does not support Remote Database Access. The remainder of the QA array contains the full data base FMP file namr, including file security code and cartridge.

Request Database Information

```
10 DIM S[10]
30 DIM Q$[14],S$[6],F$[6],L$[6]
40 DIM B$[2]
60 DATA 1,7,100,201
70 DATA "  QA:100:43;"
80 DATA "FAIL  "
120 READ M1,M7,I1,M2
130 READ Q$
140 READ F$
          .
          .
   code to open database
          .
          .
250 CALL DBINF(Q$,F$,M2,S[1],B$)
260 IF S[1]#0 THEN 720
270 IF B$#"- " THEN 750
          .
          .
          .
720 PRINT "SYSTEM ERROR = ",S[1]," ON INFO CALL."
730 PRINT "CALL A SYSTEMS ANALYST."
740 GOTO 770
750 PRINT "YOUR LEVEL DOESN'T ALLOW YOU TO ENTER A TRANSACTION."
760 PRINT "GOODBYE."
          .
          .
  code to close the database
          .
          .
```

DBINF is called in mode 201 to determine if the user has read/write access to the database or only read access. In the case that the first element of B$ is "- ", entries may be added to or deleted from the data set.

Read Entry (Forward Chain)

```
10 DIM S[10]
20 DIM Q$[14],F$[6],N$[6],P$[8],A$[10],D$[6],L$[11]
30 DATA "  QA:100:43;"
40 DATA "FAIL  "
50 DATA "PART# "
60 DATA "ASSY#,DATE;"
70 DATA 5,1
80 READ Q$,F$,N$,L$
90 READ M5,M1
          .
          .
   code to open database and
   determine chain to be read
          .
          .
500 CALL DBFND(Q$,F$,M1,S[1],N$,P$)
510 IF S[1]#0 THEN 600
520 CALL DBGET(Q$,F$,M5,S[1],P$,L$,A$,D$)
530 IF S[1]#0 THEN 680
540 PRINT P$,A$,D$
550 GOTO 520
          .
          .
          .
600 IF S[1]=156 THEN 640
610 IF S[1]#107 THEN 660
620 PRINT "NO RECORD FOR THIS PART EXISTS."
630 GOTO 200
640 PRINT "NO HISTORY EXISTS FOR THIS PART."
650 GOTO 200
660 PRINT "SYSTEM ERROR ",S[1]," ON FIND."
670 GOTO 900
680 IF S[1]=155 THEN 200
690 PRINT "SYSTEM ERROR ",S[1]," ON GET."
700 GOTO 900
```

In order to do a chained read, a call to DBFND must be done to properly initialize the chain pointers. If the status returned is 107, there is no master entry for that key value. If the status code returned is 156, the master entry exists, but there are no detail records in the chain.

Once the call to DBFND has completed successfully, the chain can be read with repeated calls to DBGET in mode 5. A status code returned of 155 indicates that there are no more records in the chain and the function is complete.

Read Entry (Serially)

```
10 DIM S[10]
20 DIM Q$[15],A$[10],L$[12],S$[6]
30 DATA "  QA:100:43;"
40 DATA "ASMBY "
[illegible]
70 READ Q$
80 READ S$
90 READ L$
100 READ M4,M0,M2
           .
           .
           .
   code to open database
           .
           .
           .
400 CALL DBGET(Q$,S$,M2,S[1],M0,L$,A$,N)
410 IF S[1]#0 THEN 600
420 PRINT A$,N
430 GOTO 400
           .
           .
           .
600 IF S[1]=12 THEN 900
610 PRINT "SYSTEM ERROR ",S[1]," ON GET."
620 GOTO 900
           .
           .
           .
```

Serial reads are performed with successive calls to DBGET in mode 2. Before beginning the serial reads, a call is made to DBGET in mode 4 (directed read) with an ARG value of zero to reset the file position pointer to the beginning of the file. A status return of 12 from the serial reads indicates that the end of the file has been reached.

Read Entry (Directed)

```
10 DIM S[10]
20 DIM Q$[15],A$[10],L$[12],S$[6]
30 DATA "  QA:100:43;"
40 DATA "ASMBY "
50 DATA "ASSY#,FAIL#;"
60 DATA 4,0,2
70 READ Q$
80 READ S$
90 READ L$
100 READ M4,M0,M2
          .
          .
          .
     code to open the
        database
          .
          .
          .
300 PRINT "WHAT RECORD DO YOU WISH?"
310 INPUT R1
320 CALL DBGET(Q$,S$,M4,S[1],R1,L$,A$,N)
330 IF S[1]#0 THEN 500
340 PRINT A$,N
350 GOTO 900
          .
          .
          .
500 IF S[1]=111 THEN 700
510 IF S[1]#114 THEN 600
520 PRINT "RECORD IS EMPTY."
530 GOTO 900
600 PRINT "SYSTEM ERROR ",S[1]," ON GET."
610 GOTO 900
700 PRINT "ILLEGAL RECORD NUMBER."
710 GOTO 900
```

The user is prompted for a relative record number and a call to DBGET is made in mode 4 to access that record. If the record number is illegal (doesn't exist) a status code of 111 is returned. If the record accessed is empty, a status code of 114 is returned.

Read Entry (Keyed)

```
10 DIM S[10]
30 DIM Q$[15],S$[6],F$[6],L$[6]
50 DIM I$[12],G$[41],Y$[10],A$[10]
60 DATA 1,7,100,201
70 DATA "  QA:100:43;"
80 DATA "FAIL  "
90 DATA "ASMBY "
100 DATA "ASSY#,FAIL#;"
110 DATA "ASSY#,DATE,PART#,FAILCD,REFDES,PNTS,EMP#;"
120 READ M1,M7,I1,M2
130 READ Q$
140 READ F$
150 READ S$
160 READ I$
170 READ G$
          .
          .
          .
   code to open database
   and determine key value
          .
          .
          .
470 CALL DBGET(Q$,S$,M7,S[1],A$,I$,Y$,F1)
480 IF S[1]#0 THEN 650
          .
          .
          .
650 IF S[1]=107 THEN 680
660 PRINT "SYSTEM ERROR = ",S[1]," ON GET."
670 GOTO 730
680 PRINT "ILLEGAL ASSEMBLY NUMBER."
690 GOTO 280
          .
          .
          .
```

Here a call to DBGET is made with a mode of 7 which performs a keyed read. If an entry with the desired key value does not exist, a status code of 107 is returned.

Add An Entry

```
10 DIM S[10]
20 DIM D$[6],P$[8],C$[4],R$[4],T$[8],E$[6]
30 DIM Q$[15],S$[6],F$[6],L$[6]
50 DIM I$[12],G$[41],Y$[10],A$[10]
60 DATA 1,7,100,201
70 DATA "  QA:100:43;"
80 DATA "FAIL  "
110 DATA "ASSY#,DATE,PART#,FAILCD,REFDES,PNTS,EMP#;"
120 READ M1,M7,I1,M2
130 READ Q$
140 READ F$
170 READ G$
          .
          .
          .
   code to open database
          .
          .
          .
280 PRINT "GET READY TO ENTER A TRANSACTION."
290 PRINT "EN ENDS THIS PROGRAM."
300 PRINT "ENTER ASSEMBLY NUMBER."
310 INPUT A$
320 IF A$[1,2]="EN" THEN 770
330 PRINT "ENTER DATA AS YYMMDD."
340 INPUT D$
350 PRINT "ENTER PART NUMBER."
360 INPUT P$
370 PRINT "ENTER FAILED CODE."
380 INPUT C$
390 PRINT "ENTER REFERENCE DESIGNATOR."
400 INPUT R$
410 PRINT "ENTER POINT VALUES (0 OR 1) WITHOUT COMMAS OR BLANKS."
420 INPUT T$
430 PRINT "ENTER EMPLOYEE NUMBER."
440 INPUT E$
450 CALL DBLCK(Q$,F$,M1,S[1])
460 IF S[1]#0 THEN 700
490 CALL DBPUT(Q$,F$,M1,S[1],G$,A$,D$,P$,C$,R$,T$,E$)
500 IF S[1]#0 THEN 600
          .
          .
          .
600 IF S[1]=107 THEN 630
610 PRINT "SYSTEM ERROR = ",S[1]," ON PUT."
620 GOTO 730
630 PRINT "ILLEGAL PART NUMBER."
```

```
640 GOTO 280
          .
          .
          .
680 CALL DBUNL(Q$,F$,M1,S[1])
690 IF S[1]=0 THEN 550
700 PRINT "SYSTEM ERROR ",S[1]," ON UNLOCK."
710 GOTO 730
```

The database must be locked before the call to DBPUT because the open had been made in mode 1. If no entry with the supplied key value exists in the associated master data set, a status code of 107 is returned. Before going to prompt the user for another transaction, the database must be unlocked. In the case of an error which causes a jump to statement 790 to close the database, an unlock is not needed since DBCLS will unlock a locked database.

Update An Entry

```
10 DIM S[10]
30 DIM Q$[15],S$[6],F$[6],L$[6]
50 DIM I$[12],G$[41],Y$[10],A$[10]
60 DATA 1,7,100,201
70 DATA "  QA:100:43;"
80 DATA "FAIL  "
90 DATA "ASMBY "
100 DATA "ASSY#,FAIL#;"
110 DATA "ASSY#,DATE,PART#,FAILCD,REFDES,PNTS,EMP#;"
120 READ M1,M7,I1,M2
130 READ Q$
140 READ F$
150 READ S$
160 READ I$
170 READ G$
          .
          .
          .
    code to open data
         base
          .
          .
          .
450 CALL DBLCK(Q$,F$,M1,S[1])
460 IF S[1]#0 THEN 700
470 CALL DBGET(Q$,S$,M7,S[1],A$,I$,Y$,F1)
480 IF S[1]#0 THEN 650
510 LET F1=F1+1
520 CALL DBUPD(Q$,S$,M1,S[1],I$,Y$,F1)
530 IF S[1]#0 THEN 580
```

```
540 CALL DBUNL(Q$,F$,M1,S[1])
550 IF S[1]=0 THEN 280
560 PRINT "SYSTEM ERROR = ",S[1]," ON UNLOCK."
570 GOTO 730
580 PRINT "SYSTEM ERROR = ",S[1]," ON UPDATE."
590 GOTO 730
          .
          .
          .
```

A database lock is done before the call to DBGET to insure that the entry is not modified between this program's get and update. After the entry has been updated, the database is unlocked and the user is prompted for another entry.

Delete An Entry

```
100 DIM S[10]
110 DIM Q$[14],P$[3],D$[3],L$[3],C$[3]
120 DATA 1,7,0
130 DATA "  QA:100:43;"
140 DATA "PROJ  "
150 DATA "CLDAT "
160 READ M1,M7,M0
170 READ Q$
180 READ P$
190 READ L$
         .
         .
         .
   code to open
   database and
   determine
   project number
   to be deleted
         .
         .
         .
300 CALL DBLCK(Q$,P$,M1,S[1])
310 IF STAT[1]#0 THEN 500
320 CALL DBGET(Q$,P$,M7,S[1],D$,L$,C$)
330 IF STAT[1]#0 THEN 600
340 CALL DBDEL(Q$,P$,M1,S[1])
350 IF STAT[1]#0 THEN 650
360 CALL DBUNL(Q$,P$,M1,S[1])
370 IF STAT=0 THEN 200
380 PRINT "SYSTEM ERROR ",S[1]," ON UNLOCK."
390 GOTO 900
          .
          .
          .
500 PRINT "SYSTEM ERROR ",S[1]," ON LOCK."
510 GOTO 900
600 IF S[1]=107 THEN 630
610 PRINT "SYSTEM ERROR ",S[1]," ON GET."
620 GOTO 900
630 PRINT "THIS PROJECT NUMBER DOESN'T EXIST."
640 GOTO 360
650 IF S[1]=113 THEN 680
660 PRINT "SYSTEM ERROR ",S[1]," ON DELETE."
670 GOTO 900
680 PRINT "TRANSACTIONS STILL EXIST FOR THIS NUMBER."
690 PRINT "IT CANNOT BE DELETED."
700 GOTO 360
          .
          .
          .
          .
          .
          .
```

The steps in deleting an entry from a database opened in mode 1 (shared mode) are lock, get, delete and unlock. If the database is opened in exclusive mode (mode 3) the lock and unlock steps are omitted.

If the entry to be deleted is in a manual master data set and there are still entries in a detail data set linked to the master entry, a status code of 113 is returned and no deletion is made.

Lock And Unlock Database

```
10 DIM S[10]
30 DIM Q$[15],S$[6],F$[6],L$[6]
60 DATA 1,7,100,201
70 DATA "  QA:100:43;"
80 DATA "FAIL  "
120 READ M1,M7,I1,M2
130 READ Q$
140 READ F$
          .
          .
          .
450 CALL DBLCK(Q$,F$,M1,S[1])
460 IF S[1]#0 THEN 700
          .
          .
          .
```

```
540 CALL DBUNL(Q$,F$,M1,S[1])
550 IF S[1]=0 THEN 280
560 PRINT "SYSTEM ERROR = ",S[1]," ON UNLOCK."
570 GOTO 730
          .
          .
          .
700 PRINT "SYSTEM ERROR = ",S[1]," ON LOCK."
710 GOTO 730
          .
          .
          .
```

If a database is opened in shared mode (mode 1), the database must be locked before updating, adding or deleting entries.

If the database is opened in mode 3 (exclusive read/write), no error will occur if an attempt to lock the database is made, but no action will be taken. A database opened in mode 8 (read only) cannot be locked and an attempt to do so will result in a status code of 134.

In all examples shown, the lock requests are made in mode 1 which is lock with wait. No return will be made to the user until the requested operation is complete. An alternative would be to request lock without wait (mode 2) in which case return is immediate with an indication of whether or not the requested operation took place.

A database need not be unlocked to be closed. If a request is made to close a locked database, an attempt to unlock will be made at that time. If the unlock is unsuccessful, the close request will return a status code of 135.

Close A Data Set Or Database

```
10 DIM S[10]
30 DIM Q$[15],S$[6],F$[6],L$[6]
60 DATA 1,7,100,201
70 DATA "  QA:100:43;"
80 DATA "FAIL  "
120 READ M1,M7,I1,M2
130 READ Q$
140 READ F$
         .
         .
         .
    code to open database
    and process transactions
         .
         .
770 CALL DBCLS(Q$,F$,M1,S[1])
780 IF S[1]=0 THEN 890
790 PRINT "SYSTEM ERROR = ",S[1]," ON CLOSE."
890 END
```

Before leaving any program that has opened the database, the database must be closed. If the database is locked, it does not need to be unlocked before attempting to close. The close operation will attempt an unlock on the database.

When the database is closed, the array which contains the database file namr is reset to its original value before the database was opened. This allows the database to be opened again in the same program without having to have reset the database name array.

The status code should be checked after a database close and reported if it is non-zero. This alerts the user that there is the possibility of some database abnormality.

Sample BASIC Program

The following is a sample BASIC program in its entirety. The program accesses the QA database and prompts the user for entries to be made in the FAIL data set.

```
10 DIM S[10]
20 DIM D$[6],P$[8],C$[4],R$[4],T$[8],E$[6]
30 DIM Q$[15],S$[6],F$[6],L$[6]
40 DIM B$[2]
50 DIM I$[12],G$[41],Y$[10],A$[10]
60 DATA 1,7,100,201
70 DATA "  QA:100:43;"
80 DATA "FAIL  "
90 DATA "ASMBY "
100 DATA "ASSY#,FAIL#;"
110 DATA "ASSY#,DATE,PART#,FAILCD,REFDES,PNTS,EMP#;"
120 READ M1,M7,I1,M2
130 READ Q$
140 READ F$
150 READ S$
160 READ I$
170 READ G$
180 PRINT "PLEASE ENTER PROPER SECURITY CODE."
190 INPUT S1
200 IF S1#I1 THEN 880
210 PRINT "PLEASE ENTER YOUR LEVEL."
220 INPUT L$
230 CALL DBOPN(Q$,L$,M1,S[1])
240 IF S[1]#0 THEN 810
250 CALL DBINF(Q$,F$,M2,S[1],B$)
260 IF S[1]#0 THEN 720
270 IF B$#"- " THEN 750
280 PRINT "GET READY TO ENTER A TRANSACTION."
290 PRINT "EN ENDS THIS PROGRAM."
300 PRINT "ENTER ASSEMBLY NUMBER."
310 INPUT A$
320 IF A$[1,2]="EN" THEN 770
330 PRINT "ENTER DATA AS YYMMDD."
340 INPUT D$
350 PRINT "ENTER PART NUMBER."
360 INPUT P$
370 PRINT "ENTER FAILED CODE."
380 INPUT C$
390 PRINT "ENTER REFERENCE DESIGNATOR."
400 INPUT R$
410 PRINT "ENTER POINT VALUES (0 OR 1) WITHOUT COMMAS OR BLANKS."
420 INPUT T$
430 PRINT "ENTER EMPLOYEE NUMBER."
```

```
440 INPUT E$
450 CALL DBLCK(Q$,F$,M1,S[1])
460 IF S[1]#0 THEN 700
470 CALL DBGET(Q$,S$,M7,S[1],A$,I$,Y$,F1)
480 IF S[1]#0 THEN 650
490 CALL DBPUT(Q$,F$,M1,S[1],G$,A$,D$,P$,C$,R$,T$,E$)
500 IF S[1]#0 THEN 600
510 LET F1=F1+1
520 CALL DBUPD(Q$,S$,M1,S[1],I$,Y$,F1)
530 IF S[1]#0 THEN 580
540 CALL DBUNL(Q$,F$,M1,S[1])
550 IF S[1]=0 THEN 280
560 PRINT "SYSTEM ERROR = ",S[1]," ON UNLOCK."
570 GOTO 730
580 PRINT "SYSTEM ERROR = ",S[1]," ON UPDATE."
590 GOTO 730
600 IF S[1]=107 THEN 630
610 PRINT "SYSTEM ERROR = ",S[1]," ON PUT."
620 GOTO 730
630 PRINT "ILLEGAL PART NUMBER."
640 GOTO 280
650 IF S[1]=107 THEN 680
660 PRINT "SYSTEM ERROR = ",S[1]," ON GET."
670 GOTO 730
680 PRINT "ILLEGAL ASSEMBLY NUMBER."
690 GOTO 280
700 PRINT "SYSTEM ERROR = ",S[1]," ON LOCK."
710 GOTO 730
720 PRINT "SYSTEM ERROR = ",S[1]," ON INFO CALL."
730 PRINT "CALL A SYSTEMS ANALYST."
740 GOTO 770
750 PRINT "YOUR LEVEL DOESN'T ALLOW YOU TO ENTER A TRANSACTION."
760 PRINT "GOODBYE."
770 CALL DBCLS(Q$,F$,M1,S[1])
780 IF S[1]=0 THEN 890
790 PRINT "SYSTEM ERROR = ",S[1]," ON CLOSE."
800 GOTO 860
810 IF S[1]#153 THEN 850
820 PRINT "YOUR LEVEL DOESN'T ALLOW ACCESS TO THE DATABASE."
830 PRINT "GOODBYE."
840 GOTO 890
850 PRINT "SYSTEM ERROR = ",S[1]," ON OPEN."
860 PRINT "CALL A SYSTEMS ANALYST."
870 GOTO 890
880 PRINT "BAD SECURITY CODE - GOODBYE."
890 END
```

Chapter 6
Maintaining The Database

There are seven utility programs available for use in the general maintenance of the database. The functions performed by these utilities are:

- loading large amounts of data into a newly created database
- copying data from a database to a backup file or magnetic tape with the option of restructuring the database upon reloading
- copying the database structure and data to a backup file or magnetic tape sector by sector
- restoring the database from a backup file to disc
- checking the amount of free space in a database
- status checking and closing a database that has been left open

The utility routines are for the use of the database manager or that individual responsible for the upkeep of the database. Creating a backup of the database using DBULD or DBSTR should be done on a regular basis to prevent loss of information through system failure or operator error.

The seven database utilities and their uses are:

DBBLD Loads actual data entries into a database. DBBLD is useful for initially storing large amounts of data into a database, or adding data entries to data already stored in a database.

DBULD Copies data entries from a database into a file on disc or

DBSTR	Copies the database root file and database entries sector by sector into a file on disc or magnetic tape, which also stores any pointers associated with the database. DBSTR is used to quickly create a back-up copy of the database that can be restored using DBRST.
DBRST	Restores a root file and associated database entries into disc files from a file on disc or magnetic tape created by the DBSTR routine. Unlike DBLOD, DBRST only restores the contents of the files. No modification of the database structure is allowed.
DBSPA	Indicates the capacity of all data sets in a database, the number of free records in each data set, and the number of records used in each data set.
RECOV	Displays the number of programs that have the database open in any access mode and allows the user to close any database inadvertently left open, thereby freeing system resources.

DBBLD was described in detail in Chapter 2, in conjunction with creating a database. However, the user should remember that DBBLD can also be useful for entering large amounts of data at any time. If bulk updates are a necessary or useful part of a database application, they can be done easily with DBBLD. See Chapter 2 for the format of the file expected by DBBLD and information on how to run the utility.

DBULD vs. DBSTR

Both DBULD and DBSTR (and their related reloading utilities) can be used for database backup. The question sometimes asked by a user is what the differences are between them and when one would use which utility. DBSTR copies the entire database, including the root file, sector by sector, doing no checking of the data or database structure. The copy will contain structure and linkage information and empty records in addition to valid data. Because of these facts, it is relatively fast to run, but uses some extra space on the output medium.

DBULD, on the other hand, copies only the data entries from the manual master and detail data sets. The data is read serially from each data set. No link or structure information is maintained. Therefore, it is somewhat slower to save a data set with DBULD than DBSTR but it takes less space on the output medium.

Though time and space are a consideration in choosing the right backup utility, there is one even more important consideration. Under certain circumstances, such as user error or system failure, the chains linking records together may be in error even though the data itself is intact. Certain records may be inaccessible with normal random read operations. However, since DBULD reads records serially, the data could be unloaded and then reloaded using DBLOD. Since DBLOD recreates the links, the reloaded file would then be totally accessible. Because DBSTR does not check for chain integrity, a file with broken chains could be stored and restored for some period of time without the broken chains being evident if they were in an infrequently accessed area of the database.

Because of this and depending on the size of the database, it may be wise to use DBULD to backup the database at regular intervals, with DBSTR backups being made more frequently in between. Of course, if any database restructuring were desired, DBULD and DBLOD would be necessary.

DBULD

DBULD stores a copy of the database information to magnetic tape or a selected disc file. Only data entries from manual master and detail data sets are unloaded. The DBULD routine is requested through a directive as follows:

```
:RU,DBULD,console,storage,root,level,abort
```

where:

console	is the console logical unit (lu). Default is the scheduling lu.
storage	is the FMP namr where the data is to be stored. Default is LU 8. The device cannot be a minicartridge unit.
root	is the FMP file namr of the database root file. If omitted, DBULD asks the user interactively for the file namr on the console lu.
level	is the highest level word defined for the database or any word if no level words were defined in the database schema. If omitted, DBULD asks the user interactively for a level word on the console lu.

abort	is AB to abort DBULD if the end of the storage device is reached; CO if the user wishes to continue under this condition. If omitted, DBULD asks the user interactively for the abort word on the console lu.

For example,

```
     :RU,DBULD,15,BCKFLE::93,QA:100:43,SECUR,AB

          or

     :RU,DBULD,15,BCKFLE::93
     ROOT FILE NAMR ?QA:100:43
     HIGHEST LEVEL WORD ?SECUR
     ABORT AT END OF STORAGE DEVICE (YES OR NO)? YES
```

After the user enters the appropriate parameters either interactively or in the RU command, DBULD proceeds to dump the database to the specified storage medium.

In some cases, an entire database can be dumped onto a single reel of tape or into a file contained on one disc cartridge. In other cases, multiple reels or files may be required. Whenever it reaches the end of a file, and the user specified not to abort, DBULD prints the following message on the console logical unit:

```
     SAVE FILE xxxxxx AS zzzzzz nn
```

where xxxxxx is the storage file name
      zzzzzz is the database name
      nn   is the number signifying the order in which the files were saved

IMAGE then asks the following:

```
     NEXT STORAGE FILE (AB TO ABORT)?
```

The user must enter an FMP file namr of the next file to use as a storage file.

When an FMP file namr is specified as the output medium, DBULD purges the existing file and creates a new one with a size that fills the remainder of the cartridge. If the end of file is reached, and the abort word was AB, DBULD aborts. If the end of file is reached and the user specified not to abort, DBULD asks the user for the name of the next storage file to be used. After storing the last entry into the output file, DBULD truncates the file to a size just large enough to hold the data it has stored in the file. A file created by DBULD cannot be transferred via a FMGR ST command. Records will be truncated and the data cannot be released.

DBULD informs the user of the completion of the dump with the message:

```
     DATABASE UNLOAD COMPLETE
```

displayed on the console. If magnetic tape was used, it is recommended that the user physically mark the various tape reels with the database name, the date of the dump, and the reel number.

DBLOD

DBLOD loads database values from a file created by DBULD into a data base. DBLOD requires that a root file exist on the disc and loads the data from the storage device according to this root file. DBLOD reinitializes the root file and all data sets prior to loading the data.

Restructuring The Database

Database information can be loaded into the same database structure, or into a different database structure. The user of DBLOD accomplishes this by creating a different database schema. The user can modify the database any one or more of the following ways:

- Rename the database by inserting a new database name in the schema.
- Assign a different security code by changing the integer number used as the security code in the schema.
- Define a new level structure by modifying the level numbers and corresponding level words in the level part of the schema.
- Change the read and write level numbers for the items in the item part of the schema.
- Define a new item structure in a detail or manual master data set that combines two or more consecutive ASCII items into one larger item. The total of the sizes of the original items should equal the size of the new item.
- Define a new item structure that takes one ASCII item in a detail or manual master data set and splits it into two or more items. The size of the original item should equal the total of the sizes of the new items.

- Define a new item structure that takes two or more consecutive numerical items or numerical item arrays into one larger numerical array. The items must be next to each other in the entry and must be of the same type (i.e.,both integer or both real). Also, the size of the array must equal the sum of the sizes of the items being combined. For example, a simple one-word integer can be combined with an immediately following ten-word integer array to form an eleven-word integer array.

- Define a new item structure that splits one numerical array into two or more consecutive numerical items or arrays. The new items must agree in type and the size of the array must equal the sum of the sizes of the items into which it is split.

- Redefine sort items in a detail data set. A sort item may be specified for each key item included in that data set.

- Change the capacity of a data set to increase or decrease the number of data entries allowed in the data set.

- Change the cartridge reference number of the root file and/or any data sets.

- Add or delete detail data sets. To add a detail data set to the data base, define a data set in the schema. This set must have a name not contained on the storage device used to load the database. No data at all will be loaded into this data set by DBLOD. To delete a data set from the database, omit the data set name from the new schema. When DBLOD loads data into the database, the routine scans the schema to find a data set name matching the name on the tape. If no data set name is found in the schema, data for that set is not loaded and is lost to the user. To load data set values into the database, the same data set name appearing on the tape must appear in the new database schema.

- Delete manual master data sets by removing the name of the set from the new database schema. If DBLOD does not find a set name in the schema corresponding to the set name on the tape, data for that set is not loaded and is lost to the user. Data for manual master data sets already existing on the tape can be reloaded merely by using that set name in the new schema. If a manual master data set is deleted, all references by existing detail data sets to that master must be omitted from the new schema.

- Delete automatic master data sets by removing the name of the set from the new database schema. If an automatic master data set is deleted, all references by existing detail data sets to that master must be omitted from the new schema.

- Add automatic master data sets by defining new sets in the database schema. If a new automatic master data set is created, care must be taken when writing the new schema that key items in detail data sets reference the proper set name. DBLOD creates the necessary item values into the automatic master data set key items at the time the database is loaded by DBLOD.

- Add manual master data sets. No data will be loaded into new manual master data sets. New manual master data sets may be related to newly-created detail data sets or may be completely independent. A new manual master data set cannot be related to existing detail data sets.

- Redefine key items in detail data sets by changing the reference to master data sets. If key items in detail data sets are changed, care must be taken that key item values contained in a detail data set already exist in the manual master data set key items.

- Add non-key items to the end of all entries of a data set. ASCII items are blank-filled and numerical items are filled with binary zeroes.

When doing any restructuring of the schema that involves splitting or combining data items, the user should take care that the total size of the old items equals the total size of the new items. When reloading data, DBLOD will lay the old data on the new item definition. If the new record is shorter than the original entry, the old entry will be truncated from the right until it fills an integral number of items, and loaded into the new entry. If the new entry is longer than the old, the old entry will fill as many complete items of the new entry as possible, with remaining items left untouched. If the old entry in its entirety is shorter than the first item of the new data set, all the data for this data set is skipped and no data is loaded into the new data set.

Running DBLOD

```
     :RU,DBLOD,console,storage,root,level,abort
```

where:

console	is the console logical unit (lu). Default is the scheduling lu.
storage	is the FMP namr where the data has been stored. Default is LU 8.

root	is the FMP file namr of the root file, according to which DBLOD will load the data. If omitted, DBLOD will ask the user for the namr interactively on the console lu.
level	is the highest level word defined for the database or any word if no level words were defined in the database schema. If omitted, DBLOD will ask the user interactively for a level word.
abort	equals AB to abort DBLOD if a duplicate data set file name is encountered. It equals CO if DBLOD is to overwrite existing files. If omitted, DBLOD asks the user interactively on the console lu for the abort word.

The user should specify AB for the abort word if it is desired that DBLOD abort if a duplicate file is encountered, thereby protecting the existing file. The user should specify any other abort word to overwrite existing files. In this case, DBLOD purges all existing files (provided the security codes match) and creates new empty ones. It then loads the data from the storage device into the new data sets.

For example:

```
     :RU,DBLOD,15,BCKFLE::93,QA:100:43,SECUR,CO

          or

     :RU,DBLOD,15,BCKFLE::93
     ROOT FILE NAMR ?QA:100:43
     HIGHEST LEVEL CODE WORD ?SECUR
     OVERWRITE EXISTING FILES (YES OR NO)?YES
```

After the user enters the appropriate parameters either interactively or in the RU command, DBLOD proceeds to load the database from the storage medium.

When the storage medium is a disc file, and DBLOD reaches the end of file, it prints the following message on the console logical unit:

```
     END OF FILE XXXXXX
     NEXT STORAGE FILE (AB TO ABORT)?
```

where XXXXXX is the name of the storage file just completed. Enter the FMP file namr of the next storage file or enter AB to abort the program.

When the load operation has completed, DBLOD informs the user with:

```
     DATABASE LOAD COMPLETED
```

displayed on the console lu.

During normal operation of DBLOD, it is recommended that the abort parameter be set to AB. This ensures that DBLOD will abort rather than overwrite an existing file that may contain important information. In this mode of operation, all the data sets used in the new schema should be manually purged prior to running DBLOD.

During database debugging operations, the abort parameter may be set to CO. DBLOD then effectively overwrites any existing data set files. It should be noted, however, that DBLOD can overwrite an existing file only if it has the same security code as the database. Therefore other existing files with duplicate names are safe as long as they have a different security code.

DBSTR

The DBSTR routine dumps a database root file and associated detail, automatic and manual master data sets (including pointers), to a magnetic tape or disc file. To request DBSTR, the user enters the directive:

```
     :RU,DBSTR,console,storage,root,level,abort
```

where:

console	is the console logical unit (lu). Default is the scheduling lu.
storage	is the FMP namr where the data is to be stored. Default is LU 8.
root	is the FMP file namr of the root file. If omitted, DBSTR asks the user interactively for the namr on the console lu.
level	is the highest level word defined for the database or any word if no level words were defined in the database schema. If omitted, DBSTR asks the user interactively for a level word on the console lu.
abort	equals AB to abort if the end of the storage device is reached; CO to continue under this condition. If omitted, DBSTR asks the user interactively for the abort word on the console lu.

For example,

```
     :RU,DBSTR,15,BCKUP:100:45,QA:100:43,SECUR,AB
         or
     :RU,DBSTR,15,BCKUP:100:45
     DATABASE NAMR ?QA:100:43
     HIGHEST LEVEL CODE WORD ?SECUR
     ABORT AT END OF STORAGE DEVICE (YES OR NO)?YES
```

If the user specified a file namr for the storage medium, DBSTR purges the existing file and creates a new one with a size that fills the remainder of the cartridge. If the end of this file is reached and the abort word equals AB, DBSTR aborts. If the end of file is reached and the abort word is CO, DBSTR asks the user for the next storage file to be used. After storing the last data into the output file, DBSTR truncates the file to a size just large enough to hold the data it has stored in the file. A file created by DBSTR cannot be transferred via a FMGR ST command. Records will be truncated and the data cannot be released.

After the database has been dumped to the storage device, DBSTR displays:

```
     DATABASE STORE COMPLETED
```

on the console lu. At this time, it is recommended that the user physically mark the various tape reels with the database name, the date of the dump, and the reel number. A file created by DBSTR cannot be transferred via a FMGR ST command. Records will be truncated and the data cannot be restored.

DBRST

The DBRST routine loads a database and associated root file into files from a backup created previously by the DBSTR routine. No modification of the root file is allowed, and the backup used must have been created by the DBSTR routine only. Since DBRST restores the root file as well as all data sets, a root file need not be available on disc to run DBRST. This is in contrast to DBLOD, which does not restore the root file and therefore, needs one available. To request DBRST, the user enters the directive:

```
     :RU,DBRST,console,storage,root,level,abort
```

where:

console	is the console logical unit (lu). Default is the scheduling lu.

storage	is the FMP namr where the data has been stored. Default is LU 8.
root	is the FMP file namr of the root file. If omitted, DBRST asks the user interactively for the namr on the console lu.
level	is the highest level word defined for the database, or any word, if no level words were defined in the database schema. If omitted, DBRST asks the user interactively on the console lu for the level word.
abort	equals AB to abort DBRST on duplicate data set file names. It equals CO to overwrite existing duplicate files. If omitted, DBRST asks the user interactively for the abort word on the console lu.

The user should specify AB for the abort word if it is desired that DBRST abort if a duplicate file namr is encountered, thereby protecting the existing file. The user should specify CO to overwrite existing files. In this case, DBRST purges all existing files (provided the security codes match) and creates new empty ones. Then it loads the data from the storage device into the new data sets.

For example,

```
:RU,DBRST,15,BCKUP:100:45,QA:100:43,SECUR,CO

     or

:RU,DBRST,15,BCKUP:100:45
ROOT FILE NAMR ?QA:100:43
HIGHEST LEVEL CODE WORD ?SECUR
OVERWRITE EXISTING FILES (YES OR NO)?YES
```

After the user enters the appropriate parameters either interactively or in the RU command, DBRST proceeds to load the database from the storage medium.

When the storage medium is a disc file and DBRST reaches the end of the file, it prints the following message on the console logical unit:

```
END OF FILE XXXXXX
NEXT STORAGE FILE (AB TO ABORT) ?
```

where XXXXXX is the name of the file just completed. Enter the name of the next file or enter AB to abort the program. If a job abort occurs during operation of DBRST because of tape reading problems, the database root file on the disc is purged. To conserve disc space, it is suggested that the user manually purge the remainder of the data sets.

DBRST informs the user of completion by displaying the message:

```
     DATABASE RESTORE COMPLETED
```

on the console lu.

During normal operation of DBRST, it is recommended that the abort parameter be set to AB. This ensures that DBRST will abort rather than overwrite an existing file that may contain important information. In this mode of operation, all the data sets used in the schema should be manually purged prior to running DBRST.

During database debugging operations, the abort parameter may be set to CO. DBRST then effectively overwrites any existing data set files. It should be noted, however, that DBRST can overwrite an existing file only if it has the same security code as the database. Therefore, other existing files with duplicate names are safe as long as they have a different security code.

DBSPA

The database space utility, DBSPA, indicates the capacity of all data sets, the number of free records for the data set and also indicates how many records are being used in the data set. This information can

If root and/or level are omitted, DBSPA will ask the user to input the the root file namr and/or the level code word. For example:

```
     :RU,DBSPA,15,6,QA:100:43,SECUR

          or

     :RU,DBSPA,15,6
     ROOT FILE NAMR ?QA:100:43
     LEVEL CODE WORD ?SECUR
```

After the user enters the appropriate parameters either interactively or in the RU command, DBSPA will print out a listing of all data sets for the database, capacity, records used, free records, and the discrepancy. If the level code word supplied by the user is less than the highest level word defined for the database, DBSPA will report on only those data sets to which the user has access. An example is:

```
IMAGE/1000 DATABASE SPACE UTILITY


DATA SET NAME  CAPACITY  FREE RECORDS  RECORDS USED  DIFFERENCE
-------------  --------  ------------  ------------  ----------

   ASMBY            409           397            12           0
   PART            4001          3996             5           0
   CODE             997           990             7           0
   DATEF            367           362             5           0
   REF              997           988             9           0

   EMPL             263           262             1           0
   PROJ             101            85            16           0
   FAIL            1009           998            11           0
   TIME             367           367             0           0

 END DBSPA
```

An example of a possibly bad database is:

```
IMAGE/1000 DATABASE SPACE UTILITY

DATA SET NAME  CAPACITY  FREE RECORDS  RECORDS USED  DIFFERENCE
-------------  --------  ------------  ------------  ----------

   ASMBY            409           397            12           0
   PART            4001          3996             5           0
   CODE             997           990             7           0
   DATEF            367           364             5         - 2
   REF              997           988             9           0

   EMPL             263           262             1           0
   PROJ             101            85            16           0
   FAIL            1009           998            11           0
   TIME             367           367             0           0

 DATABASE MAY NOT BE GOOD

 END DBSPA
```

RECOV

RECOV is an operator invocable utility which displays information about all databases currently known to IMAGE and lets the operator remove erroneous table entries that may have been left open by an IMAGE program that did not properly close a database.

Information about the databases in the system is stored in a table in the control program DBCOP. In addition, information about programs remotely accessing databases is stored in a table called RD.TB, the RDBAP copy scheduling table. For every non-null entry in the table, RECOV displays the database name, the cartridge number, the open mode of the database, and the names of all the programs which have the database open. If any of the programs accessing the database are remote database access program copies, RECOV also displays the master program's name and node number.

After displaying the information, RECOV asks the operator if there is a program to be cleaned up after. If the response is YES, RECOV asks for its name and determines whether the program name exists in the coordinating table. If the operator has specified a program whose name was not found, RECOV issues an error message and displays the coordinating table again. If the program name is valid, RECOV simulates a closure for each database that the program has open. If the simulated closure is successful, RECOV reports

```
DONE.
```

If the closure of a database reduces the user count to zero, RECOV reports

```
DATABASE <dbname> RELEASED.
```

If the program could not be removed from the coordinating table, RECOV reports the message

```
CLOSURE UNSUCCESSFUL.
```

After each clean-up, RECOV displays the new version of the coordinating table and again asks the operator if there is another program to be cleaned up after. This process is repeated as long as the user keeps responding YES. If the coordinating table is unobtainable, RECOV prints a message and terminates.

To run RECOV, enter the following:

```
:RU,RECOV,console,list,node
```

[illegible]

console	is the console logical unit (lu). Default is the scheduling lu. If console is a non-interactive device, the coordinating table will be printed once on the list device and RECOV will terminate.
list	is the list lu. Default is the console lu.
node	is the node number at which the operator resides in a DS network. Default is the local node. If RECOV is run from a DS/1000 node, all lus specified must be system, rather than session lus. To execute RECOV remotely, the user must be sure to run REMAT and execute the switch (SW) command. RECOV should be executed using the run with wait (RW) command.

The following is sample RECOV table output:

1) Without RDBA in the system:

```
**************************************
DB NAME      CART #    MODE    OPEN TO
--------------------------------------

  QA           43        1       PAR21
                                 TRANS

  DB2          100       3       SRC27

**************************************
```

2) With RDBA in the system:

```
*************************************************************
DB NAME       CART #    MODE    OPEN TO    MASTER     NODE
------------------------------------------------------------

  QA           43        1       PAR21
                                 TRANS

  DB2          100       3       RDB02        MAST1      1

-------------------------------------------------------------
```

Chapter 7
Advanced Topics

The purpose of this chapter is to provide information which will aid the advanced user in understanding the structure and functioning of IMAGE. The user should be aware that all IMAGE data sets should be accessed only by IMAGE routines and never directly by the user. Any attempt to alter an IMAGE data set could violate the integrity of the database.

Structural Elements Of The Database

An IMAGE database consists of data sets plus a file containing an internal description of the database called a Root File. In addition to the information in the Root File, there is also structural information stored in each record in the data sets. The information stored in each record is called the media record and its purpose is to indicate if the record contains an entry or not, and if so, chain pointers to related entries. A pointer is a two word entity containing the relative record number of another record in the set. Each entry in a chain has a pair of pointers, one pointing to the predecessor and one pointing to the successor within the chain. The first entry in a chain has a zero for a backward pointer and the last entry in the chain has a zero as a forward pointer.

Detail Data Sets

Detail data set records can belong to one of two chains; the free record chain or a data chain. All unused or free records in a detail data set are chained together by forward pointers only, so that when an entry is to be added to the set, an unoccupied record can be found quickly.

Data chains are chains of detail entries that share a value for a key item. Each detail entry is contained in as many chains as there are keys in that set. The beginning of a chain is located by accessing that master entry which also has that given key value. The technique for doing so is discussed in the following paragraphs.

An entry's location within a chain depends on whether or not that chain is sorted. In unsorted chains, the order of the entries is chronological - that is, a new entry is placed at the end of the chain. If the chain is sorted, however, a new entry would be inserted in the chain at a location determined by the value of its sort item.

The format of the media record for a detail data set is shown in Figure 7-1.

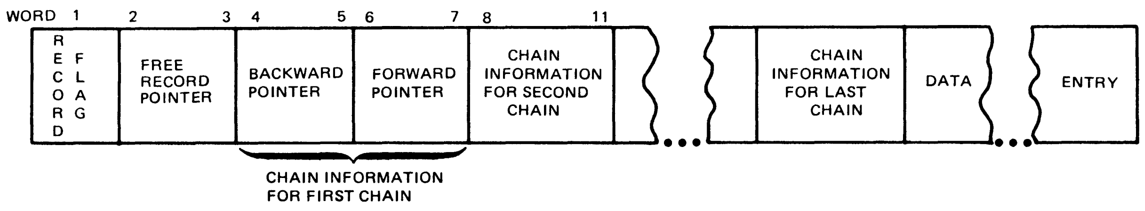

WORD 1 2 3 4 5 6 7 8 11
RECORD FLAG
FREE RECORD POINTER
BACKWARD POINTER
FORWARD POINTER
CHAIN INFORMATION FOR SECOND CHAIN
CHAIN INFORMATION FOR LAST CHAIN
DATA
ENTRY
CHAIN INFORMATION FOR FIRST CHAIN

Figure 7-1. Detail Set Media Record

The record flag is 0 if the record is empty or greater than zero if the record contains an entry. If the record is empty, the next two words will contain the pointer to the next free record. The next 4n words are forward and backward pointers in the data chains, one set for each path to the set. Starting in word 4n+4 is the actual data entry. The size of the detail data set record is calculated as (3+4n+m) words where n is the number of paths to the data set, and m is the size of the data entry.

Master Data Sets

When an entry is placed into a master data set, the key item value is converted to a relative record number by the use of a hashing algorithm. This technique is discussed later in this chapter. The first entry to hash to a given location is called a primary entry. If a subsequent entry has the same relative record number, it is called a synonym. A synonym is placed in the next available empty record, and is connected to the associated primary entry via a synonym chain. Synonym chain pointers consist of both backward and forward pointers. Records containing synonyms are called secondary entries.

IMAGE locates the first or last entry in a chain by using a chain head. The chain head for a particular chain is stored with the entry in the corresponding master data set whose key item value is the same as the detail key item value defining the chain. Each chain head is six words long and consists of three doubleword entries; the length of the chain, the record number of the first entry in the chain and the record number of the last entry in the chain.

The format of the media record for a master data set is shown in Figure 7-2.

The first word is entry type and is 0 if the record is empty, 1 if the record contains a primary entry and -1 if the record contains a synonym. The next four words are the synonym

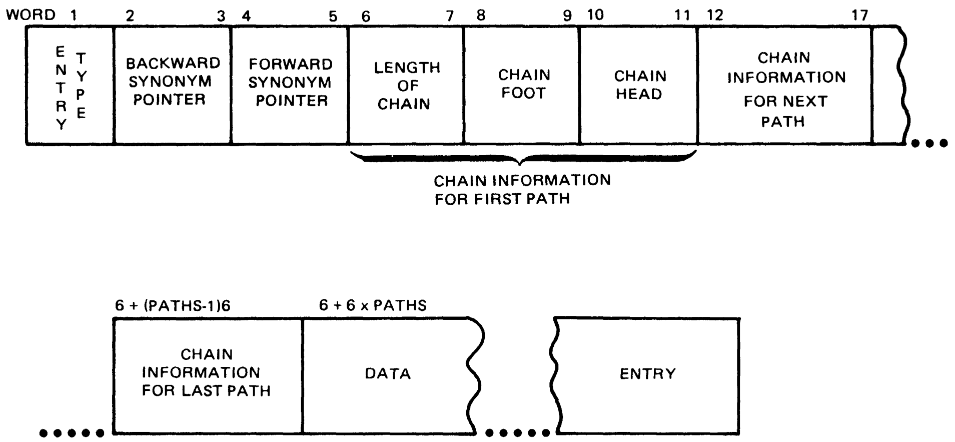


Figure 7-2. Master Set Media Record

chain pointers. Words 6 through 6n+5 are chain heads, one six word entry for each path from this data set. The actual data entry begins in word 6n+6. The size of the record, then, is (5+6n+m) words where n is the number of paths in this set, and m is the size of the data entry.

Root File

The Root File contains a description of the database. It is stored in an FMP file whose file namr is that given as the database namr in the schema. When a database is opened, the Root File is read into the user's partition in the area from the end of the user's program to the end of the partition (or if EMA is being used, the space used is from the end of the program to the beginning of the MSEG). When the Root File is read into the user partition, some of the information becomes dynamic and the Root File is referred to as the Run Table.

The Root File consists of several tables. These tables are the:

- Database Control Block (DBCB) - contains general database information and pointers to other tables.
- Data Item Table (DIT) - contains one entry for each item in the data base.
- Data Set Control Block Table (DSCBT) - contains one entry for each data set with general set description information.
- Data Set Info Table (DSIT) - contains an entry for each set. Each set entry contains a record definition table and a path table. A record definition table has item numbers for each item in the set. The path table has an entry for each path in the set.
- Sort Table (ST) - contains data item numbers and data set numbers arranged in alphabetical order of their corresponding item or set names. The purpose of the sort table to facilitate searching the DIT, DSCBT and DSIT.
- Free Record Table (FRT) - contains a free record count for each data set and a pointer to the first record of the free record chain for detail data sets.

The detailed structures of each of these tables are shown in the following sections. The user is reminded that these tables are shown for information only and the user should never attempt to modify anything in IMAGE's internal structures.

Database Control Block

The Database Control Block has two different structures depending on whether it is on disc as part of the Root File or in memory as part of the Run Table. The format shown in Figure 7-3 is the Run Table format. Fields marked with an asterisk are filled in at open time. In addition, words 15-59 in the Root File are reserved for the level code words which are not needed for the Run Table. Some of that space is re-used; the remainder is currently unused.

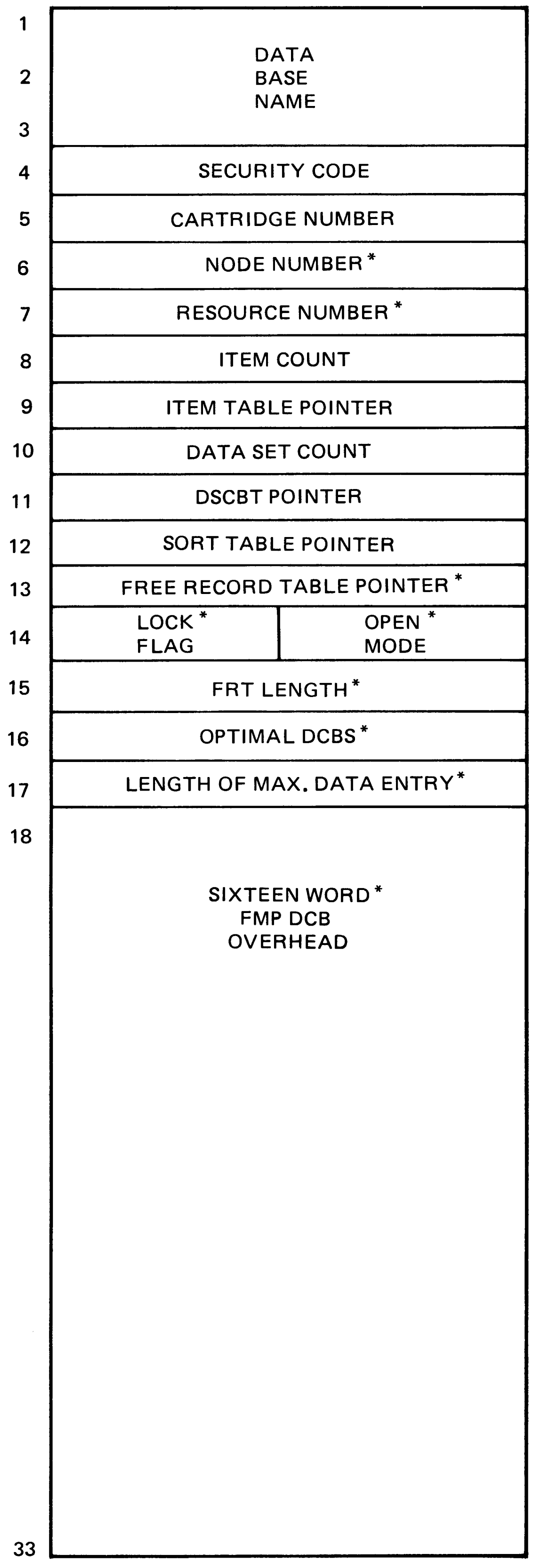


Figure 7-3. Database Control Block

Data Item Table

The Data Item Table contains one seven word entry for each item in the database. The format of the entry is shown in Figure 7-4. The entries are ordered by data item number with the first entry representing data item 1. The data set count is the count of the number of sets in which this item appears. The data set number is the number of the first data set in which the item occurs. The set number gives a starting point in the data set info table to use to begin a search and the set count gives an indication of when the search is complete. The W and/or R bits are set at open time if the user has write and/or read access to this item.

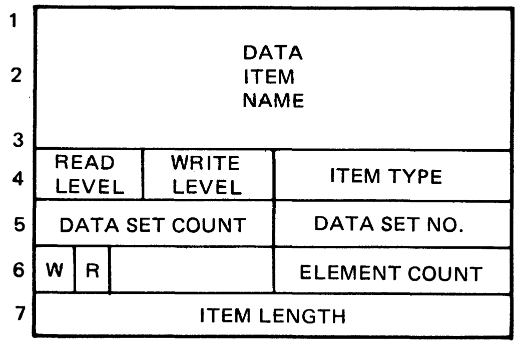


Figure 7-4. Data Item Table

Data Set Control Block Table

The Data Set Control Block Table contains one seventeen word entry for each data set in the database. The format is shown in Figure 7-5. The W bit is set if the user can add or delete entries in this set. The R bit is set if the user may not add or delete entries but can read at least one item in the set. The RA bit is set if the user may read all items in the data set. W, R, and RA bits are set at open time. The set type is 0 for automatic master, 1 for manual master and 2 for detail. The current search item number is the number of the master set's key item. If the set is a detail, this value is zero.

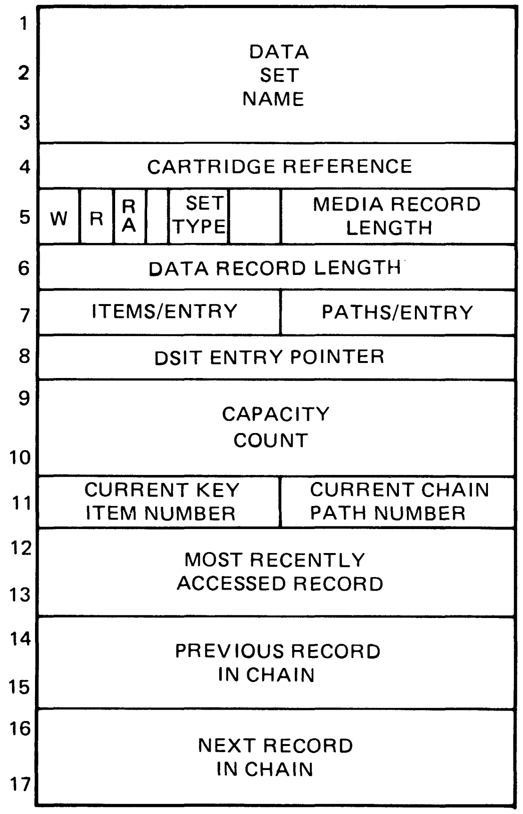


Figure 7-5. Data Set Control Block Table

Data Set Info Table

The Data Set Info Table contains a record definition table and a path table for each data set. The tables are ordered in descending set number order, with the path table for a given set following after the associated record definition table. The format of these two tables is shown in Figure 7-6. The data set number in the path table is a detail set number if the set is a master or a master set number if the set is a detail.

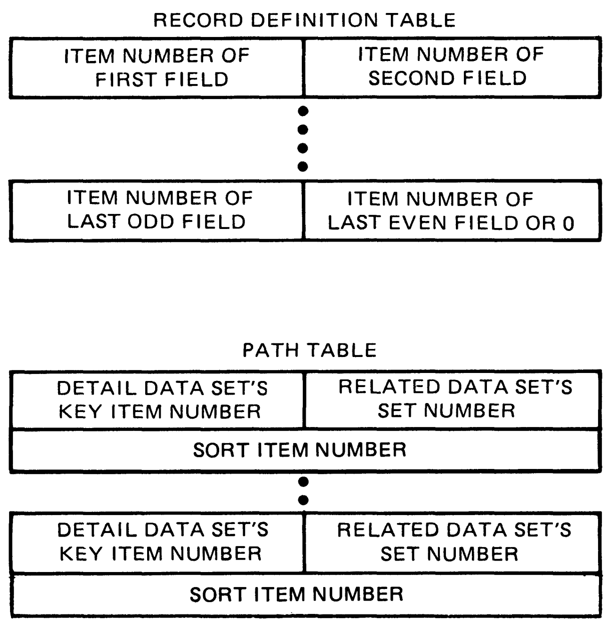

RECORD DEFINITION TABLE
ITEM NUMBER OF FIRST FIELD
ITEM NUMBER OF SECOND FIELD
ITEM NUMBER OF LAST ODD FIELD
ITEM NUMBER OF LAST EVEN FIELD OR 0
PATH TABLE
DETAIL DATA SET'S KEY ITEM NUMBER
RELATED DATA SET'S SET NUMBER
SORT ITEM NUMBER
DETAIL DATA SET'S KEY ITEM NUMBER
RELATED DATA SET'S SET NUMBER
SORT ITEM NUMBER

Figure 7-6. Data Set Info Table

Sort Table

The purpose of the Sort Table is to allow binary searching for a set or item entry. The Sort Table is a list of data item numbers and data set numbers arranged in alphabetic order of the corresponding names. The format is shown in Figure 7-7.

Free Record Table

The Free Record Table contains one four word entry for each data set in the database. The entry contains a doubleword free record count and, if this is a detail set, a doubleword record number of the first record in the free record chain. If this is a master set, the third and fourth words of the entry are zero. Although this table is part of the Run Table, it is also updated on disc whenever the values in it change. The format of the table entry is shown in Figure 7-8.

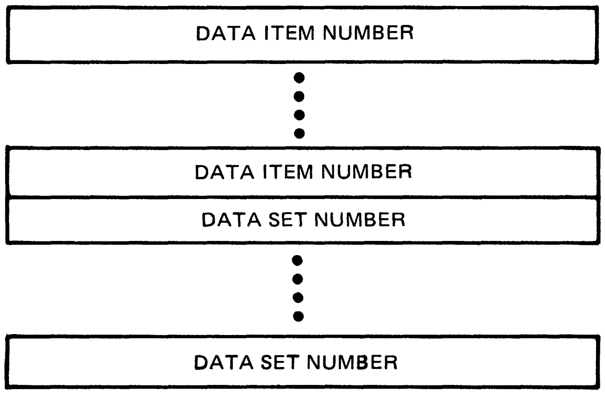


Figure 7-7. Sort Table

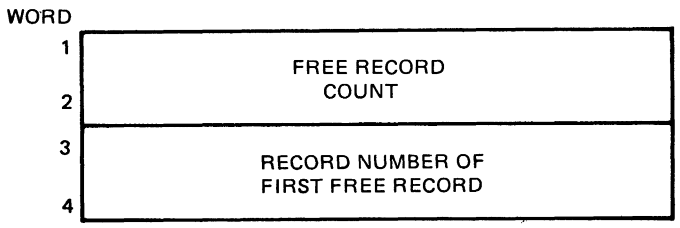


Figure 7-8. Free Record Table Entry

Space Allocation

When calculating the extra space needed from the end of the program to the end of the partition (or the size of n in the PASCAL $IMAGE n$ compiler option), three areas need to be considered. One is space for the Run Table, the second is space for FMP Data Control Blocks (DCBs), and the third is the record buffer.

The size of the Root File is printed at the end of the listing produced by DBDS. It takes the following form:

```
     ROOT FILE:    00423 WORDS,   00007 BLOCKS
```

The number of words given is the space required for the Run Table. The following discussion shows how this size is calculated. First consider the following variables.

Table 7-1. Run Table Size Calculation Variables

Variable	Definition
i	Number of data items in the database
s	Number of data sets in the database
e	Number of data items in entry for a given data set
p	Number of data paths in a given data set

The calculations for each table and the total are as follows.

Table 7-2. Run Table Size Calculations

Table	Size
DBCB	59
DIT	7i
DSCBT	17s
DSIT	$\sum(e+1)/2 + \sum 2p$
ST	i + s
FRT	4s
TOTAL	$59 + 8i + 22s + \sum(e+1)/2 + \sum 2p$

As an example, let us look at the QA Database and calculate the amount of space necessary for the Run Table. The number of items in the database is 14 and the number of sets is 9. The calculation for the size of the DSIT is as follows.

s	e	$\frac{e+1}{2}$	p	2p	$\frac{e+1}{2} + 2p$
1	2	1	1	2	3
2	1	1	1	2	3
3	1	1	1	2	3
4	1	1	2	4	5
5	1	1	1	2	3
6	1	1	3	6	7
7	4	2	1	2	4
8	8	4	7	14	18
9	4	2	3	6	8
					54

Figure 7-9. Calculation of Space for QA Database

So the total size required for the run table is 59 + 8*14 + 22*9 + 54 = 423 words.

The Record Buffer is the size of the largest entry in any database the user currently has open. No matter how many data sets and databases are in use, there is only one record buffer allocated.

The remaining space is used to allocate DCBs. The minimum amount of space needed for DCBs is 272 words. The maximum amount of space allocated is one DCB for each path in the set in the database which has the greatest number of paths plus one additional DCB. All DCBs are 272 words long.

Remote Database Access

Remote Database Access (RDBA) enables use of a database which does not reside at the local node. This is accomplished with DS through user-written programs or by running QUERY remotely. The following is an example of remote QUERY:

```
:RU,REMAT
$SW,40,,DS
#RW,QUERY,1,1,1,,45



    IMAGE/1000 QUERY



NEXT? DATA-BASE=QA:100:43;
LEVEL = ?
MODE = ? 1;
NEXT? SELECT-FILE=SEL;
NEXT? EXIT;
#EX
 $END REMAT
:
```

To make a user-written program access a remote database, two things must be done: The DS node number should be placed in the first word of the IBASE parameter before the DBOPN is done, and the program should be loaded with $DBMS2 or $DBMS3 as a library. Refer to the IMAGE/1000 Configuration Guide for further information on loading programs which access a remote database.

REMOTE DATABASE ACCESS RUN TABLE SIZE

If a user is using DS to access a database residing at a remote node, the complete Run Table resides at the same node as the database. RDBAP, a remote database access program, acts as the user program at the remote site and the complete Run Table resides in its partition. In the user's partition at the local site, only abbreviated versions of the DBCB, DSCBT, DIT, and ST plus a Record Buffer are allocated. The total amount of space required is 19 words plus 6 words per data set plus 5 words per item plus the number of words in the largest entry in the database plus 127 words or one word per item in the database, whichever is larger.

Technique For Keyed Read

IMAGE employs two distinct methods of calculating relative word numbers in master data sets. The first method applies to master data sets with key items of type I or R. The low-order 31 bits of the R type key or the 16 bits of the I type key are zero filled from the left to form a positive 32-bit doubleword value. This doubleword value is reduced modulo the data set capacity and incremented by one to form a relative record number. This is the record number of the primary entry.

The second method of record number calculation applies to master data sets with key items of type X. In this case, the entire key item value regardless of its length, is used to obtain a positive doubleword value. The value is reduced modulo the data set capacity and incremented by one to form a relative record number. The algorithm used to obtain the doubleword intermediate value attempts to approximate a uniform distribution of relative record numbers in the master data set, regardless of the bias of the master data set key item values.

The intent of the two algorithms is to spread master entries as uniformly as possible throughout the record space of the data set. This uniform spread reduces the number of synonyms occurring in the master data set.

Migrating Secondaries

In some cases, a secondary entry of a master data set is automatically moved to a storage location other than the one originally assigned. This most often occurs when a new master data entry is assigned as a primary entry to a record occupied by a secondary entry. By definition, the secondary entry is a synonym to some other primary entry resident at their common primary address. Thus, the new entry represents the beginning of a new synonym chain. To accomodate this new chain the secondary entry is moved to an alternate secondary address and the new entry is added to the data set as a primary entry. This move and the necessary linkage and chain head maintenance is done automatically.

A move can also occur when the primary entry of a synonym chain having one or more secondary entries is deleted. Since retrieval of each entry occurs through a synonym chain, each synonym chain must have a primary entry. To maintain the integrity of a synonym chain, IMAGE always moves the first secondary entry to the primary location. The former first secondary entry is now the primary entry for the chain and the record formerly containing the secondary entry is now empty.

Reducing Synonyms

It should be clear that accessing a record located in a synonym chain will take longer than accessing a primary entry. Locating a synonym requires hashing the key value to determine the primary entry and following through the chain, comparing the key item value with that of each entry in the chain. It is desirable from a performance standpoint, therefore, to minimize the number of synonyms in a master data set.

To determine the number of synonyms in a data set, the user should write a program which will determine the current number of synonyms in the database. Once it has been determined that the percentage of synonyms is too high, there are three steps that can be taken to reduce them. The first is to make the capacity of the data set a prime number. In almost all cases, changing the capacity from a non-prime number to a prime number reduces the number of synonyms. Another factor which can influence the number of synonyms in a data set is the number of empty records in the set. The total number of records used should not exceed 80% of the capacity, and ideally should be even less. As the data set is filled to beyond 80% of capacity, the database can be restructured using DBULD and DBLOD to increase the capacity of the data set.

If applying both of these solutions does not bring the synonyms to an acceptable number it may be advantageous to write a specialized hash routine. In some cases, the database application produces key values that are not spread over the whole range of possible values, but are concentrated in a subset of values. In this case, the user should consider writing a hash routine that fits that particular application and substituting it for the hash routine used by IMAGE.

Replacing The Standard Hash Routine

Should a user choose to substitute a new hash routine for the standard, the substitution should be done in two steps. The first step is to test the routine, using an analyzer program such as was used to determine the original number of synonyms. This analysis will indicate if the new routine will reduce the number of synonyms sufficiently. If not, refine the routine.

Once the new hash routine is sufficiently refined, all application programs that access the database must be reloaded, with the new hash routine relocated along with the application program. This includes user-written programs and all IMAGE utilities that use a hashed read to access the database. These include QUERY, DBBLD, DBLOD, and RDBAP. It is important to insure that all programs hashing to the database use the same hash routine. Failure to do so will result in unpredictable results and possible corruption of the database.

Format Of The Hash Routine

If the user is to supply a substitute hash routine it must be a routine that returns a positive doubleword value in the A and B registers. In FORTRAN, this would be a real-valued function. IMAGE will take this value modulo the capacity, incremented by 1 as the relative record number. The calling parameters are the length of the value to be hashed and the beginning address of the key. The entry point must be named "HASH".

Hash Routine Example

The following is an example of a user written hash routine.

```
FTN4,L,C
C
C          THIS FUNCTION RETURNS A PSEUDO RANDOM NUMBER BETWEEN
C          THE VALUES OF ZERO AND (2**31)-1 WHICH CAN BE USED AS
C          A HASHED RECORD NUMBER IN AN IMAGE DATABASE
C          APPLICATION.
C
      REAL FUNCTION HASH(LEN,IARG)
C
C          LEN = LENGTH IN WORDS OF IARG
C          IARG = BEGINNING WORD OF THE KEY
C
      DIMENSION IARG(1),IDUM(3)
      DOUBLE PRECISION D,DF
      EQUIVALENCE (IDUM,D),(DF,RET),(RET,IFRST)
C
C          INITIALIZE DOUBLE PRECISION NUMBERS:
C          DF (MANTISSA = .1, EXPONENT = 0)
C
      DF=.1D0
C
C          USE ENTIRE KEY ITEM VALUE, SCRAMBLING THE WORDS INTO
C          A DOUBLE PRECISION REAL, ADDING IN THE FRACTIONAL
C          PORTION OF PI, AND MULTIPLYING THEM TOGETHER.
C
      DO 5 I=1,LEN
        DF=DF*(IARG(I)+.1415926535D0)
5     CONTINUE
C
C          EXTRACT THE MANTISSA WORDS OF CALCULATED
C          VALUE, USE ONLY 31 DATA BITS, AND NOT THE
C          SIGN BIT.
C
      IFRST=IAND(IFRST,77777B)
      HASH=RET
      RETURN
      END
      END$
```

QUERY Select File

The select file is used by QUERY to store relative record numbers as a result of a FIND command. REPORT and UPDATE commands then reference this select file to determine the records to use in their operations.

It is also possible for the user to perform a FIND while in QUERY, exit from QUERY and then access the contents of the select file from a user program. This user program can then use the relative record numbers to perform directed reads on the data set in question.

The select file has additional information in its header which facilitates its use. Each select file contains a sixteen word header formatted as follows.

Word	Contents
1-9	Database File Namr
10	Data Set Number
11-12	Number of records selected
13-16	Reserved
17-n	Doubleword integers representing relative record numbers

Files named in a SELECT-FILE= command can be created prior to execution of the command, but need not be. If the file named as a select file doesn't exist at the time the SELECT-FILE= command is executed, QUERY will create the file as a type 1 file, six blocks long. If the select file is created by the user it must be a type 1 file of at least three blocks long. Under certain circumstances, six blocks may not be sufficient. If QUERY overflows the select file, an error message will be reported. The user must then create a select file that is larger than three blocks.

DBCOP

DBCOP (Database Coordinating Program) is the program used to coordinate all database activity in the system. It contains information about each database open in the system, including the number of users open to each database. Shared access of databases is coordinated through DBCOP in the following manner.

When a request is made to open a database, DBOPN schedules DBCOP with parameters that indicate what database is to be opened and in what mode it will be opened. DBCOP checks its table and determines if such an open is possible and returns that information to DBOPN. If the database is available, DBOPN completes its open processing and reschedules DBCOP (when open processing is finished) to have DBCOP alter its table to reflect the latest open. As part of the close processing, DBCOP is again scheduled so that it can decrement the user count for a database just as it incremented it as a result of the DBOPN operation.

DBCOP is also used by RECOV to get the information to display the coordinating table. The information currently available in DBCOP in the coordinating table is displayed by RECOV. If the user wishes to clean up after a program, RECOV passes this information to DBCOP, and DBCOP adjusts its table accordingly.

In order for DBCOP to be available for scheduling, it must have an ID segment. If it is loaded on-line and saved as a program file, it must be RP'ed before a data base can be accessed. Offing DBCOP while any IMAGE programs are running will cause valuable information about the state of the system's databases to be lost. If DBCOP is offed while databases are open, all IMAGE processing should be brought to a conclusion in an orderly manner, a fresh copy of DBCOP should be made available, and database activity resumed.

Generation And Loading Considerations

For information about generation and on-line loading of IMAGE programs, see the IMAGE/1000 Configuration Guide.

DS Considerations

In general, if using Remote Database operations, time-out values should be higher than for normal DS operations. If a large number of DS08 (time-out) errors should occur, when Remote Database Access is being performed, a large time-out value could help the problem.

Users should not perform a call to DBLCK with wait to a remote database. If the database is already locked, a DS08 error will occur.

Appendix A
HP Character Set For Computer Systems

Effect of Control key *

000-037B		040-077B		100-137B		140-177B	

b4	b3	b2	b1	b7 b6 b5 →	0 0 0	0 0 1	0 1 0	0 1 1	1 0 0	1 0 1	1 1 0	1 1 1
BITS				COLUMN → / ROW ↓	0	1	2	3	4	5	6	7
0	0	0	0	0	NUL	DLE	SP	0	@	P	`	p
0	0	0	1	1	SOH	DC1	!	1	A	Q	a	q
0	0	1	0	2	STX	DC2	"	2	B	R	b	r
0	0	1	1	3	ETX	DC3	#	3	C	S	c	s
0	1	0	0	4	EOT	DC4	$	4	D	T	d	t
0	1	0	1	5	ENQ	NAK	%	5	E	U	e	u
0	1	1	0	6	ACK	SYN	&	6	F	V	f	v
0	1	1	1	7	BEL	ETB	'	7	G	W	g	w
1	0	0	0	8	BS	CAN	(	8	H	X	h	x
1	0	0	1	9	HT	EM	)	9	I	Y	i	y
1	0	1	0	10	LF	SUB	*	:	J	Z	j	z
1	0	1	1	11	VT	ESC	+	;	K	[	k	{
1	1	0	0	12	FF	FS	,	<	L	\	l	¦
1	1	0	1	13	CR	GS	—	=	M	]	m	}
1	1	1	0	14	SO	RS	.	>	N	^	n	~
1	1	1	1	15	SI	US	/	?	O	_	o	DEL

32 CONTROL CODES

Upshifted Lower Case

64 CHARACTER SET

96 CHARACTER SET

128 CHARACTER SET

EXAMPLE: The representation for the character "K" (column 4, row 11) is.

```
          b7 b6 b5 b4 b3 b2 b1
BINARY    1  0  0  1  0  1  1
OCTAL     1     1        3
```

* Depressing the Control key while typing an upper case letter produces the corresponding control code on most terminals. For example, Control-H is a backspace.

9206- 1A

HEWLETT-PACKARD CHARACTER SET FOR COMPUTER SYSTEMS

This table shows HP's implementation of ANS X3.4-1968 (USASCII) and ANS X3.32-1973. Some devices may substitute alternate characters from those shown in this chart (for example, Line Drawing Set or Scandanavian font). Consult the manual for your device.

The left and right byte columns show the octal patterns in a 16 bit word when the character occupies bits 8 to 14 (left byte) or 0 to 6 (right byte) and the rest of the bits are zero. To find the pattern of two characters in the same word, add the two values. For example, "AB" produces the octal pattern 040502. (The parity bits are zero in this chart.)

The octal values 0 through 37 and 177 are control codes. The octal values 40 through 176 are character codes.

Decimal Value	Octal Values		Mnemonic	Graphic[1]	Meaning
	Left Byte	Right Byte			
0	000000	000000	NUL	␀	Null
1	000400	000001	SOH	␁	Start of Heading
2	001000	000002	STX	␂	Start of Text
3	001400	000003	ETX	␃	End of Text
4	002000	000004	EOT	␄	End of Transmission
5	002400	000005	ENQ	␅	Enquiry
6	003000	000006	ACK	␆	Acknowledge
7	003400	000007	BEL	␇	Bell, Attention Signal
8	004000	000010	BS	␈	Backspace
9	004400	000011	HT	␉	Horizontal Tabulation
10	005000	000012	LF	␊	Line Feed
11	005400	000013	VT	␋	Vertical Tabulation
12	006000	000014	FF	␌	Form Feed
13	006400	000015	CR	␍	Carriage Return
14	007000	000016	SO	␎	Shift Out } Alternate
15	007400	000017	SI	␏	Shift In } Character Set
16	010000	000020	DLE	␐	Data Link Escape
17	010400	000021	DC1	␑	Device Control 1 (X-ON)
18	011000	000022	DC2	␒	Device Control 2 (TAPE)
19	011400	000023	DC3	␓	Device Control 3 (X-OFF)
20	012000	000024	DC4	␔	Device Control 4 ($\overline{\text{TAPE}}$)
21	012400	000025	NAK	␕	Negative Acknowledge
22	013000	000026	SYN	␖	Synchronous Idle
23	013400	000027	ETB	␗	End of Transmission Block
24	014000	000030	CAN	␘	Cancel
25	014400	000031	EM	␙	End of Medium
26	015000	000032	SUB	␚	Substitute
27	015400	000033	ESC	␛	Escape[2]
28	016000	000034	FS	␜	File Separator
29	016400	000035	GS	␝	Group Separator
30	017000	000036	RS	␞	Record Separator
31	017400	000037	US	␟	Unit Separator
127	077400	000177	DEL	▒	Delete, Rubout[3]

Decimal Value	Octal Values		Character	Meaning
	Left Byte	Right Byte		
32	020000	000040		Space, Blank
33	020400	000041	!	Exclamation Point
34	021000	000042	"	Quotation Mark
35	021400	000043	#	Number Sign, Pound Sign
36	022000	000044	$	Dollar Sign
37	022400	000045	%	Percent
38	023000	000046	&	Ampersand, And Sign
39	023400	000047	'	Apostrophe, Acute Accent
40	024000	000050	(	Left (opening) Parenthesis
41	024400	000051	)	Right (closing) Parenthesis
42	025000	000052	*	Asterisk, Star
43	025400	000053	+	Plus
44	026000	000054	,	Comma, Cedilla
45	026400	000055	–	Hyphen, Minus, Dash
46	027000	000056	.	Period, Decimal Point
47	027400	000057	/	Slash, Slant
48	030000	000060	0	
49	030400	000061	1	
50	031000	000062	2	
51	031400	000063	3	
52	032000	000064	4	
53	032400	000065	5	} Digits, Numbers
54	033000	000066	6	
55	033400	000067	7	
56	034000	000070	8	
57	034400	000071	9	
58	035000	000072	:	Colon
59	035400	000073	;	Semicolon
60	036000	000074	<	Less Than
61	036400	000075	=	Equals
62	037000	000076	>	Greater Than
63	037400	000077	?	Question Mark

Decimal Value	Octal Values		Character	Meaning
	Left Byte	Right Byte		
64	040000	000100	@	Commercial At
65	040400	000101	A	
66	041000	000102	B	
67	041400	000103	C	
68	042000	000104	D	
69	042400	000105	E	
70	043000	000106	F	
71	043400	000107	G	
72	044000	000110	H	
73	044400	000111	I	
74	045000	000112	J	
75	045400	000113	K	
76	046000	000114	L	
77	046400	000115	M	
78	047000	000116	N	Upper Case Alphabet, Capital Letters
79	047400	000117	O	
80	050000	000120	P	
81	050400	000121	Q	
82	051000	000122	R	
83	051400	000123	S	
84	052000	000124	T	
85	052400	000125	U	
86	053000	000126	V	
87	053400	000127	W	
88	054000	000130	X	
89	054400	000131	Y	
90	055000	000132	Z	
91	055400	000133	[	Left (opening) Bracket
92	056000	000134	\	Backslash, Reverse Slant
93	056400	000135	]	Right (closing) Bracket
94	057000	000136	^ ↑	Caret, Circumflex; Up Arrow[4]
95	057400	000137	_ ←	Underline; Back Arrow[4]

Decimal Value	Octal Values		Character	Meaning
	Left Byte	Right Byte		
96	060000	000140	`	Grave Accent[5]
97	060400	000141	a	
98	061000	000142	b	
99	061400	000143	c	
100	062000	000144	d	
101	062400	000145	e	
102	063000	000146	f	
103	063400	000147	g	
104	064000	000150	h	
105	064400	000151	i	
106	065000	000152	j	
107	065400	000153	k	
108	066000	000154	l	
109	066400	000155	m	
110	067000	000156	n	Lower Case Letters[5]
111	067400	000157	o	
112	070000	000160	p	
113	070400	000161	q	
114	071000	000162	r	
115	071400	000163	s	
116	072000	000164	t	
117	072400	000165	u	
118	073000	000166	v	
119	073400	000167	w	
120	074000	000170	x	
121	074400	000171	y	
122	075000	000172	z	
123	075400	000173	{	Left (opening) Brace[5]
124	076000	000174	¦	Vertical Line[5]
125	076400	000175	}	Right (closing) Brace[5]
126	077000	000176	~	Tilde, Overline[5]

9206- 1C

Notes: [1]This is the standard display representation. The software and hardware in your system determine if the control code is displayed, executed, or ignored. Some devices display all control codes as "||", "@", or space.

[2]Escape is the first character of a special control sequence. For example, ESC followed by "J" clears the display on a 2640 terminal.

[3]Delete may be displayed as "__", "@", or space.

[4]Normally, the caret and underline are displayed. Some devices substitute the up arrow and back arrow.

[5]Some devices upshift lower case letters and symbols (` through ~) to the corresponding upper case character (@ through ^). For example, the left brace would be converted to a left bracket.

Appendix B
IMAGE Numbered Error Messages

Image Numbered Error Messages

The following table is a list of the numbered error messages returned from the library subroutine, as well as the IMAGE utilties. In addition to those that are generated by IMAGE, an error return may also be a value less than 100. This is a File Manager or DS/1000 error code which IMAGE passes to the user. Before passing an FMP error value, IMAGE removes the minus sign. For more information about error codes less than 100, see the RTE-L Programmer's Reference Manual. DS error codes are signified by a -1 (minus one) in the status word. The 2nd and 3rd words of the status array contain either an ASCII DS error code or a zero followed by an integer DS error code.

Table B-1. Numbered Error Messages

Number	Meaning
-159	Internal IMAGE error, or corrupt operation index in DS message buffer
-158	No class number available for RDBAP copy
-156	No room in RD.TB for another entry, or no ID segments left for an RDBAP copy, or schedule of RDBAP aborted, or too many pending requests on RDBAP class number
-144	RDBAP copy was aborted
-1	DS error. The 2nd and 3rd words of the status array contain either an ASCII DS error code or a 0 followed by an integer DS error code
0-99	Absolute value of negative FMP error code
100	Illegal data set reference
101	Illegal item reference
102	Data item specified is not a key or sort item.

103	Illegal database parameter, or database not open to user
104	The mode of operation for the database (specified in the DBOPN call) is insufficient for the type of of access requested in a subroutine call.
105	Detail data set is full; no empty record space exists
106	Master data set is full; no empty record space exists
107	A master data set does not contain an entry for a key item value
108	A request has been directed to an automatic master data set
110	A key data item value already exists in a manual master data set
111	Path has not been initialized for a chain read, or record number in a directed read is illegal
112	User has attempted to change the value of a key or sort item
113	User has attempted to delete a manual master data set entry that still has links to non-empty detail data set chains
114	Record accessed is empty
115	The mode specified by the user is invalid
116	File specified by the IBASE parameter is not a root file
117	The security code given does not match the security code for the database being accessed
118	Data set is not write enabled for the user's access level
119	The root file for the database being accessed does not exist
120	The data set being accessed is not a detail data set

121	The detail data set being accessed is not linked to any master data sets
122	A chain read cannot be performed
123	Data set being accessed is not a master data set
124	Illegal DBINF request
125	Invalid data set or item reference used in DBINF request
128	Not enough space for IMAGE data buffers (solution: make program smaller, segment program or run program in a larger partition)
129	Root file is opened exclusively to another user
131	No activity table space available (more than 20 different databases are open in the system) or the database is already open to seven users.
132	No resource number available
134	An attempt is being made to lock a database that has been opened in mode = 8
135	Database is locked to user and attempts to unlock it have failed
136	Database is locked to another user (returned on lock without wait request)
137	Illegal resource number usage
140	Unable to schedule DBCOP
150	Database already open to user
152	Database open to another user in an incompatible mode
153	User has no access to any item or set in the data base with his given level
154	Database corrupt - bad data path pointers in a data set
155	Beginning of chain or end of chain found on a chain read
156	Detail data set does not contain any entries along the specified chain
157	No current record for data set

159	User attempting to UPDATE, PUT or DELETE a record with an open mode of 1 without first locking the database
160	IMAGE data structures are corrupt (could be caused by the structures being overwritten or by DBBUF not residing in the main segment)
161	An internal subroutine passed DBCOP an illegal function code (could be caused by a user writing over a constant in an IMAGE subroutine)
162	Missing parameter in a library subroutine call
201	User passed DBBLD an illegal database name
202	User passed DBBLD an incorrect security code
203	No data appears in the DBBLD input file between the first record and the $END record
204	A data entry in the DBBLD input file is too short or omitted
205	A $SET: or $END record is missing or too many fields exist for a data entry in a DBBLD input file
206	A non-numeric character or invalid number appeared in an integer data field in a DBBLD input file
207	A non-numeric character appeared in a numeric data field in a DBBLD input file
208	An error has occurred during entry into a database from a DBBLD input file and DBBLD cannot purge the database
209	A physical end-of-file mark was encountered by DBBLD while processing an input file
210	The logical unit specified in the RUN command for one of the utilities (DBRST, DBSTR, DBLOD, DBUNL) was illegal
211	The user has failed to give the correct level word to DBRST, DBSTR, DBLOD or DBUNL
212	The input file presented to DBLOD or DBRST is of an illegal format
213	The user has failed to give the proper security code to DBRST, DBSTR, DBUNL or DBLOD

214	The root file and associated sets data are on different subchannels during operation of DBSTR or DBRST
215	Either the file containing the database root file or the file containing the database cannot be found during operation of the DBSTR or DBRST routines
216	Either the root file size or the database file size on the disc is not the same size as the root file and database stored on the back-up file
217	The DBULD routine has discovered no data in the database to unload
218	Illegal mode or column parameter
219	Not enough space to load root file
220	DBBLD has encountered an I/O abort
230	An attempt is being made to write to a tape that has no write ring
231	An attempt is being made to access a tape on a drive that is off-line
235	The utility is aborting because it reached the end of the FMP storage file without reaching the end of the database and the user did not specify to continue
236	The utility is aborting because it reached the end of tape before reaching the end of the data base and the user did not specify to continue
241	The lu specified as the console lu is not interactive
242	The user mounted a reel with an incorrect reel number or specified a disc file out of sequence
243	The root file name specified does not match the root file name on the storage device
244	The user specified the wrong cartridge to restore the database. (DBRST must load the database back onto the same cartridge from which it came.)
247	The program was terminated by the user entering the operator BR[EAK] command
248	An illegal word was entered as an abort option in the run string

302	Too many names in a NAME-LIST
303	Invalid name in NAME-LIST
304	The type or length of a variable in VALUE-LIST does not match the type or length of its corresponding item in NAME-LIST
305	Variable missing in VALUE-LIST
306	Invalid relative record number (non-numeric)
310	Illegal IBASE parameter on a DBOPN call (does not start with two blanks)
324	Illegal DBINF request
7777	A catastrophic error has occurred in a utility - reload and rerun the program

DBDS Error Messages

The following table lists the error messages possible from DBDS.

Table B-2. DBDS Error Messages

Message	Meaning
ALL PATHS TO MASTER ARE NOT DEFINED	Not enough detail data sets defined a path to a master data set.
AT LEAST ONE ITEM MUST BE DEFINED	The database must have a minimum of one item defined.
AUTOMATIC MASTER MUST HAVE KEY ITEM ONLY	An automatic master may have only one item defined and it must be a path item.
AUTO-MASTERS MUST HAVE AT LEAST 1 PATH	An automatic master data set must define a path item with a path count of at least 1.

AUTO-MASTER NEEDS MORE WRITE CAPABILITY	An automatic master's write level must be equal to or less than its related detail data set's key item. (This is not true of manual masters.)
BAD CAPACITY COUNT	The capacity count was not between 1 and 2**31-1.
BAD READ LEVEL	The read level was not between 1 and 15.
BAD TERMINATOR - ';' EXPECTED	A semi-colon was expected. DBDS scans until it finds a semi-colon before it continues processing.
BAD TYPE DESIGNATOR	An item has been defined as other than I1, R2, or Xn or a data set has been defined as other than Manual, Automatic, or Detail.
BAD WRITE LEVEL	The write level on an item definition is not between 1 and 15 or is less than the read level.
'BEGIN DATABASE' EXPECTED	The 'BEGIN DATABASE:' command was not found.
CANNOT OPEN SCHEMA FILE	DBDS cannot open the schema file because the wrong name was entered, the disc is not mounted or there was a disc error.
'CAPACITY:' EXPECTED	The 'CAPACITY:' command was expected.
DATABASE HAS NO DATA SETS	No data sets were defined for the database.
DATA SET FILES CREATED	All the data set files were successfully created.
DATA SET MUST HAVE AN ITEM	Every data set must contain at least one data item.
DUPLICATE ITEM NAME	The item name was already defined.

DUPLICATE LEVEL WORD	The level word was already defined.
DUPLICATE SET NAME	The data set name was already used.
END DATABASE DEFINITION	DBDS has terminated.
'END.' EXPECTED	An 'END.' command was not found.
END OF FILE ENCOUNTERED	An end of file was unexpectedly found in the schema input file.
'ENTRY:' EXPECTED	The 'ENTRY:' command was expected.
ENTRY TOO BIG	A data entry is larger than 4096 bytes.
FMP ERROR - ALL DATA SETS NOT CREATED	There was an FMP error while trying to create a data set. The data set may have already existed or the disc was not mounted. Examine the disc to see which data sets were created, correct the disc error and retry DBDS.
FMP ERROR - XXXX ON FILE YYYY	FMP error on file YYYY.
ILLEGAL CARTRIDGE NUMBER	The cartridge number is not an integer between 1 and 32767 or is not two ASCII characters.
ILLEGAL CONTROL OPTION	A command found in the '$CONTROL:' list was not legal.
ILLEGAL LEVEL WORD	An illegal character was found in the level word.
ILLEGAL NAME	Some portion of the database namr or some portion of the set namr is not legal.
ILLEGAL SECURITY CODE	The security code is not an integer between 1 and 32767 or is not two ASCII characters.

ILLEGAL SEPARATOR	A comma, colon or parenthesis was expected.
ILLEGAL SORT DESIGNATOR	Sort designator is not a name, item is not defined in ITEMS: part of the schema, or sort item is the same as the key item for the path.
INCOMPLETE NAME	A database namr must have a security code.
ITEM NAME EXPECTED	A name was expected in an item definition.
ITEM TOO LONG	An item has been defined with too many elements in its array or the n in an Xn item is longer than 255.
'ITEMS:' EXPECTED	The 'ITEMS:' command was expected.
KEY ITEMS NOT OF SAME LENGTH OR TYPE	A key item in a detail data set is not the same type or length as the master's key item.
LEVEL NUMBER ALREADY DEFINED	The level number was already defined.
LEVEL OUT OF RANGE	A level number was not between 1 and 15.
'LEVELS:' EXPECTED	The 'LEVELS:' command was not found.
MASTER MUST HAVE A PATH	All master data sets, both manual and automatic must have a key item defined. The path count for a manual master may be zero.
MAX ERRORS-SCHEMA PROCESSING TERMINATED	The number of errors is equal to the requested error count specified in the '$CONTROL:' option list by the 'ERRORS=' command.
MISSING PROGRAM SEGMENTS	A segment to DBDS was not loaded.

'NAME:' EXPECTED	The 'NAME:' command was expected.
NON-NUMERIC PATH COUNT	The path count in a master data set is not an integer.
NOT ENOUGH MEMORY TO CREATE ROOT FILE	DBDS was not loaded into a large enough partition. Increase the partition size using the SZ command.
PATH ITEM CANNOT BE AN ARRAY	An item which was an array was defined as a path item.
ROOT FILE CREATED	The root file was successfully created.
ROOT FILE NOT CREATED	The root file was not created because of the 'NOROOT' or 'NOSET' option in the '$CONTROL:' list.
ROOT FILE NOT CREATED - SCHEMA ERRORS	There were errors in the schema so the root file and all the data sets were not created.
ROOT FILE OVERHEAD RECORD ERROR	There was an error writing the overhead record to the root file. Check for a possible disc error.
SEARCH ITEM ALREADY DEFINED	The master already has a path item defined. There can only be one path item per master.
'SETS:' EXPECTED	The 'SETS:' command was not found.
SORT ITEM MUST BE SIMPLE	Sort item cannot have been defined in the ITEMS: part of the schema with an element count greater than one.
SORT ITEM NOT DEFINED IN SET	Sort item does not appear in the ENTRY: part of the set definition.
THE FOLLOWING ITEM(S) ARE UNUSED	The items that are listed were not used in any sets.

TOO MANY DATA ITEMS	More than 255 data items have been defined in the database or more than 127 data items in the data set.
TOO MANY DATA SETS	More than 50 data sets were defined.
TOO MANY ELEMENTS	More than 255 elements were defined in an item definition.
TOO MANY PATHS	More than 16 paths were defined in the master or detail data set or more paths from a detail were defined to a master than were allotted to the master.
TOTAL ITEM LENGTH NOT INTEGRAL WORDS	An ASCII item defined without elements must have an even character count or the element count times the character count must be even.
UNDEFINED ITEM REFERENCED	An item was used in the set part that was not defined in the item part.
UNDEFINED SET REFERENCED	A set name was used in a detail data set's path definition that was not defined previously in the set part.
UNEXPECTED 'END.'	An 'END.' command was unexpectedly encountered.
WARNING - SECURITY CODE IGNORED	A security code in a set namr is ignored. Data sets have the same security code as the database. The error count is not increased as this is only a warning.
'$CONTROL:' EXPECTED	The '$CONTROL:' command was expected.

DBBLD Error Messages

The following table lists the error messages possible from DBBLD.

Table B-3. DBBLD Error Messages

Message	Meaning
DBBLD LIST I/O ERROR	An I/O request to the list device has failed

DBSPA Error Messages

The following table lists the error messages possible from DBSPA.

Table B-4. DBSPA Error Messages

Message	Meaning
/DBSPA - ERROR <XXX> ON DATA SET	IMAGE error while accessing data set.
/DBSPA - ERROR <XXX> ON DBOPN	IMAGE error returned from DBOPN call.
/DBSPA - ERROR ON INPUT <XXXX>	I/O error from input LU.
/DBSPA - ERROR ON OUTPUT <XXXX>	I/O error to list device.
/DBSPA - ILLEGAL OR MISSING ROOT FILE NAMR	
/DBSPA - UNABLE TO LOCK OUPUT LU	LU already locked.
/DBSPA - UNABLE TO OBTAIN INFORMATION ON DATA SETS	IMAGE error from DBINF call.

RECOV Error Messages

The following table lists the error messages possible from RECOV.

Table B-5. RECOV Error Messages

Message	Meaning
BAD PROGRAM NAME	Program name misspelled or not in coordinating table.
CLEAN UP UNSUCCESSFUL	DS error or program does not have database opened.
RECOV INPUT ERR	I/O error from input LU.
RECOV OUTPUT ERR	I/O error to list device.
UNABLE TO LOCK LIST LU	LU already locked.
UNABLE TO OBTAIN CURRENT DATA BASE INFORMATION	Not enough SAM or DBCOP not loaded.

QUERY Error Messages

The following two tables list the error messages output by QUERY. Table B-6 is an alphabetical list of the messages put out by QUERY. Table B-7 is a translation table from IMAGE numbered errors to QUERY error messages.

Table B-6. QUERY Messages

Message	Meaning
A MASTER ENTRY WITH KEY VALUE EXISTS	This is a translation of IMAGE error 110. See Table B-1 for more information.
BAD DATA SET	The data set specified does not exist in the database being processd, or the level word used in opening the database does not give the user access to this data set.

BAD DATA SET OR DATA ITEM IN DBINF CALL	This is a translation of IMAGE error 125. See Table B-1 for more information.
BAD OR MISSING SEGMENT	A segment needed by QUERY does not have an ID segment.
BAD TRACK IN WORK AREA	A track in the work area of the disc is faulty or not available. Current command terminates with no change to the database or the contents of a procedure file.
BATCH FILE ERROR	The batch file specified in the XEQ= command cannot be opened.
BEGINNING OR END OF CHAIN ENCOUNTERED	This is a translation of IMAGE error 155. See Table B-1 for more information.
BREAK REQUESTED	The user entered an RTE BR[EAK] command while QUERY was processing a report command.
CANNOT ALTER THE VALUE OF AN ITEM	This is a translation of IMAGE error 112. See Table B-1 for more information.
CANNOT EDIT REAL VALUES	A DETAIL, GROUP, or TOTAL statement contains a real data item and an edit mask number. Real data items cannot be edited.
CANNOT PERFORM A CHAIN READ	This is a translation of IMAGE error 122. See Table B-1 for more information.
CARTRIDGE LOCKED	This is a translation of FMP error 13. See the RTE-IV Programmer's Reference Manual for more information.
COMMAND TABLE OVERFLOW	A REPORT procedure contains more than 100 statements. Alter the REPORT procedure to contain less than 100 statements.
CONFLICT IN SST DEFINITION	This is a translation of FMP error 39. See the RTE-IV Programmer's Reference Manual for more information.

CONSTANT LITERAL HAS EDIT OPTION	A REPORT command statement containing a literal constant value (as opposed to a data item name) references an edit mask in an edit statement. The REPORT command terminates. Alter the REPORT procedure to delete the edit mask reference or change the constant to a data item name.
CONTROL BREAK INCONSISTENCY	The level numbers of a TOTAL, SORT, or GROUP statement do not match properly. Alter the REPORT to maintain control break consistency.
DATABASE ALREADY LOCKED TO ANOTHER USER	This is a translation of IMAGE error 136. See Table B-1 for more information.
DATABASE ALREADY OPEN	This is a translation of IMAGE error 150. See Table B-1 for more information.
DATABASE CANNOT BE CLOSED DUE TO LOCK	This is a translation of IMAGE error 135. See Table B-1 for more information.
DATABASE CORRUPT - BAD CHAIN POINTER	This is a translation of IMAGE error 154. See Table B-1 for more information.
DATABASE INACESSIBLE WITH THAT LEVEL	The level code word entered allows no access to the database.
DATABASE MUST BE LOCKED	This is a translation of IMAGE error 159. See Table B-1 for more information.
DATABASE OPEN EXCLUSIVELY	This is a translation of IMAGE error 129. See Table B-1 for more information.
DATABASE NOT DECLARED	A QUERY command was issued prior to declaration of the database. Type a DATA-BASE= command, then re-enter the QUERY command.
DATABASE NOT ENABLED FOR LOCKING	This is a translation of IMAGE error 134. See Table B-1 for more information.

DATABASE NOT OPENED	This is a translation of IMAGE error 103. See Table B-1 for more information.
DATABASE RN IS BEING USED ILLEGALLY	This is a translation of IMAGE error 137. See Table B-1 for more information.
DATA ITEM NOT MEMBER OF SET	The data item specified is not a member of the data set being processed.
DATA ITEM VALUE TOO LONG	A data item value exceeds 255 characters.
DATA SET FOR OPERATION MUST BE A DETAIL	This is a translation of IMAGE error 120. See Table B-1 for more information.
DATA SET FOR OPERATION MUST BE A MASTER	This is a translation of IMAGE error 123. See Table B-1 for more information.
DATA SET IS NOT WRITE ENABLED	This is a translation of IMAGE error 118. See Table B-1 for more information.
DETAIL CHAIN IS EMPTY	This is a translation of IMAGE error 156. See Table B-1 for more information.
DETAIL DATA SET HAS NO PATHS	This is a translation of IMAGE error 121. See Table B-1 for more information.
DETAIL DATA SET IS FULL	This is a translation of IMAGE error 105. See Table B-1 for more information.
DCB NOT OPEN	This is a translation of FMP error 11. See the RTE-IV Programmer's Reference Manual for more information.
DETAIL LEVEL XX IS MISSING	DETAIL statements using level numbers must be contiguous starting at 1. A DETAIL statement using level 1 may be defined without defining a DETAIL statement with no level number.

DEVICE I/O ABORTED	QUERY had an I/O abort error. This message appears only on the log device.
DIRECTORY FULL	This is a translation of FMP error 14. See the RTE-IV Programmer's Reference Manual for more information.
DISC CARTRIDGE NOT FOUND	This is a translation of FMP error 32. See the RTE-IV Programmer's Reference Manual for more information.
DISC DOWN	This is a translation of FMP error 1. See the RTE-IV Programmer's Reference Manual for more information.
DUPLICATE DATA ITEM IN SORT STATEMENTS	Two or more SORT statements in a REPORT procedure contain the same data item name. Alter the REPORT to eliminate the double reference.
DUPLICATE EDIT STATEMENT	An edit mask number was defined twice. Alter the REPORT procedure to eliminate the double reference.
DUPLICATE NAME	This is a translation of FMP error 2. See the RTE-IV Programmer's Reference Manual for more information.
EDIT MASK TABLE OVERFLOW	A REPORT command contains more than ten edit statements. Alter the REPORT command to eliminate the excess edit masks.
XXXXXX ENTRIES FOUND	This message informs the user of the number of entries selected by the FIND command.
ERROR NO. XXX	An error occurred which is not in either of QUERY's translation tables. If the error number is greater than 99, see Table B-1 for more information. If the error is less than 100, see the FMP File Manager Manual. Note that QUERY removes the minus sign from FMP error codes before reporting them.
FILE CURRENTLY OPEN OR EXCLUSIVE OR LOCK REJECT	This is a translation of FMP error 8. See the RTE-IV Programmer's Reference Manual for more information.

FILE NOT FOUND	This is a translation of FMP error 6. See the RTE-IV Programmer's Reference Manual for more information.
FILE SPECIFIED IS NOT A ROOT FILE	This is a translation of IMAGE error 116. See Table B-1 for more information.
FIND EXPECTED	A procedure file was used in a FIND command that did not include the FIND command. The probable cause is an error in creating the procedure file or supplying the wrong procedure file name.
FIND PROCEDURE TOO LONG	A FIND procedure contains more than 50 logical relationships.
GREATER THAN 255 EXTENTS	This is a translation of FMP error 46. See the RTE-IV Programmer's Reference Manual for more information.
ILLEGAL ACCESS TO DATA SET	The user does not have write access to the data set. Re-open the data set using a higher level code word.
ILLEGAL ACCESS TO LU	This is a translation of FMP error 20. See the RTE-IV Programmer's Reference Manual for more information.
ILLEGAL ACCESS TO SYSTEM DISC	This is a translation of FMP error 19. See the RTE-IV Programmer's Reference Manual for more information.
ILLEGAL CONTROL OPERAND	This message indicates an internal error in QUERY's tables. This message appears only on the log device.
ILLEGAL DATA ITEM NAME	The data item name specified does not appear in the schema for the database being accessed or the level code word used when opening the database did not grant a high enough capability to access this item.

ILLEGAL DATA SET NAME	The data set name specified does not appear in the database being processed, or the level code word used when opening the database did not grant enough capability to access this database.
ILLEGAL DESTINATION LU	This is a translation of FMP error 21. See the RTE-IV Programmer's Reference Manual for more information.
ILLEGAL INPUT PARAMETER	The input file declared in the RU,QUERY command cannot be opened.
ILLEGAL LIST PARAMETER	The list file declared in the RU,QUERY command cannot be opened.
ILLEGAL LOCK REQUEST	There was an error when QUERY tried to lock the list lu.
ILLEGAL LOG PARAMETER	The log file declared in the RU,QUERY command cannot be opened.
ILLEGAL LU	This is a translation of FMP error 18. See the RTE-IV Programmer's Reference Manual for more information.
ILLEGAL NAME	This is a translation of FMP error 15. See the RTE-IV Programmer's Reference Manual for more information.
ILLEGAL OPTION	The option parameter in the RU,QUERY string did not begin with the letters "EC".
ILLEGAL PATH MODIFICATION	The user tried to change a key item value in an UPDATE REPLACE command.
ILLEGAL READ OR WRITE ON TYPE 0	This is a translation of FMP error 17. See the RTE-IV Programmer's Reference Manual for more information.
ILLEGAL SECURITY CODE	This is a translation of IMAGE error 117. See Table B-1 for more information.

ILLEGAL SELECT-FILE SIZE OR TYPE	The select-file specified by the user existed, but is not a type 1 file, or is not at least six sectors long.
ILLEGAL SORT VALUE MODIFICATION	The user tried to change a sort item value in an UPDATE REPLACE command.
ILLEGAL TERMINATOR	An item value being entered must end with a semicolon. This error could also occur if too many values are entered for an array item.
ILLEGAL TYPE OR SIZE = 0	This is a translation of FMP error 16. See the RTE-IV Programmer's Reference Manual for more information.
INCONSISTENCY BETWEEN OPTIONS AND EDIT STATEMENTS	A REPORT command specified an edit mask that did not exist, or an edit mask was used on a data item that is a type real.
INPUT FILE ERROR - YYYYYYYYY	A File Manager error occurred when the RU,QUERY command was entered. YYYYYYYYY is a description of the File Manager error that occurred. This message appears only on the log device.
INPUT MUST BE CONTAINED WITHIN MULTIPLE LINES OF 72 CHARACTERS	The user entered more than 72 characters per line. QUERY ignores the command.
INPUT TOO LONG	The command entered exceeded 896 characters. Retype the command in a shorter form.
INSUFFICIENT WORK AREA FOR SORT	The area on the disc reserved for sorting data entries is not large enough to sort all of the desired entries. Use the FIND command to shorten the number of data entries found, reduce the number of items sorted or increase the number of system tracks available or simply try again later when the system is less busy.

INTERNAL BUFFERS ARE CORRUPT	This is a translation of IMAGE error 160. See Table B-1 for more information.
INTERNAL DBCOP ERROR	This is a translation of IMAGE error 161. See Table B-1 for more information.
INTERNAL IMAGE CALL IS INVALID	This is a translation of IMAGE error 162. See Table B-1 for more information.
INVALID COMMAND	QUERY does not recognize the command.
INVALID DATA ITEM NAME OR NUMBER	This is a translation of IMAGE error 101. See Table B-1 for more information.
INVALID DATA SET NAME OR NUMBER	This is a translation of IMAGE error 100. See Table B-1 for more information.
INVALID DATA ITEM VALUE OR TERMINATOR	A value is not enclosed in quotes or a comma or logical connector is missing.
INVALID DBINF MODE	This is a translation of IMAGE error 124. See Table B-1 for more information.
INVALID LOGICAL CONNECTOR XXXXXX	QUERY anticipates an AND, OR, or END and does not recognize the characters specified by XXXXXX.
INVALID MODE	The mode parameter entered was not an acceptable open mode value.
INVALID # OF VALUES FOR RELATIONAL OPERATOR	More than one value follows an IGT, INGT, ILT, or INLT where only one value is allowed.
INVALID OR MISSING SEARCH ITEM	This is a translation of IMAGE error 102. See Table B-1 for more information.
INVALID PROCEDURE NAME	The name of a procedure used in a command is invalid. Retype the command using the proper name.

INVALID REQUEST	The level code word or the open access mode are not legal.
INVALID SECURITY CODE OR ILLEGAL WRITE ON LU	This is a translation of FMP error 7. See the RTE-IV Programmer's Reference Manual for more information.
ITEM NOT MEMBER OF DATA SET OR DATA SET NOT SPECIFIED	A qualified data item is not a member of the data set specified in the FIND command or a data item is a member of more than one data set and the desired data set is not specified.
LIST FILE ERROR	The list file specified in the LIST= command could not be opened.
LU NOT FOUND IN SST	This is a translation of FMP error 40. See the RTE-IV Programmer's Reference Manual for more information.
MASTER DATA SET IS FULL	This is a translation of IMAGE error 106. See Table B-1 for more information.
MORE THAN 5 FIELDS ARE BEING TOTALED	TOTAL statements in a REPORT command reference more than five different data items.
MUST ENTER PATH ITEM VALUE	A user must enter values for path (or key) items in an UPDATE ADD command.
MUST ENTER SORT ITEM VALUE	A user must enter values for sort items in an UPDATE ADD command.
NEXT?	This is an indication that QUERY is ready for another command.
NO ACCESS TO DATA SETS	The user does not have access to any data set in the database.
NO CURRENT CHAIN OR BAD RECORD NUMBER	This is a translation of IMAGE error 111. See Table B-1 for more information.
NO CURRENT RECORD FOR OPERATION	This is a translation of IMAGE error 157. See Table B-1 for more information.

NO MASTER ENTRY WITH KEY ITEM VALUE	This is a translation of IMAGE error 107. See Table B-1 for more information.
NO RESOURCE NUMBER AVAILABLE	This is a translation of IMAGE error 132. See Table B-1 for more information.
NO ROOM FOR DATABASE ENTRY IN DBCOP	This is a translation of IMAGE error 131. See Table B-1 for more information.
NON-NUMERIC IN INTEGER VALUE	A non-digit character was entered as a value for a type integer data item.
NON-NUMERIC IN REAL OR INTEGER VALUE	The user entered a non-numeric digit when a real or integer value was expected.
NON-NUMERIC IN REAL VALUE	A non-digit character was entered as a value for a type real data item.
NOT ENOUGH ROOM ON DISC CARTRIDGE	This is a translation of FMP error 33. See the RTE-IV Programmer's Reference Manual for more information.
OPEN MODE DOES NOT ALLOW THAT OPERATION	This is a translation of IMAGE error 104. See Table B-1 for more information.
QUERY - OUTPUT FILE ERROR	QUERY had a non-recoverable file I/O error while trying to write to the list file or the log file. This message appears only on the log device.
PARTITION IS TOO SMALL	This is a translation of IMAGE error 128. See Table B-1 for more information.
PROCEDURE NAME XXXXXX NOT FOUND	The specified procedure does not exist.
QUERY/1000 READY	QUERY prints this message to inform the user that QUERY is executing.

READ OR WRITE TO A RECORD NOT WRITTEN	This is a translation of FMP error 5. See the RTE-IV Programmer's Reference Manual for more information.
RECORD HAS NOT YET BEEN FOUND	The user attempted to UPDATE REPLACE, UPDATE DELETE, or REPORT without first using a FIND command.
RECORD IS EMPTY	This is a translation of IMAGE error 114. See Table B-1 for more information.
RELATIONAL OPERATOR INVALID	The user entered a relational operator that QUERY does not recognize. Only IS, IE, INE, IGT, INGT, ILT, INLT and ISNOT are recognized.
RELEASE TRACKS	More values are specified in a FIND command than can be stored in available system tracks. Either reduce the number of FIND data item values or make more system tracks available or retry when the system is less busy.
REPORT CANNOT BE GENERATED DUE TO ERRORS	The REPORT command generated contains errors.
REPORT EXPECTED	A procedure file was used in a REPORT command that did not include a REPORT statement.
REQUEST DIRECTED AT AN AUTOMATIC MASTER	This is a translation of IMAGE error 108. See Table B-1 for more information.
RETRIEVAL FROM MORE THAN ONE DATA SET	An attempt was made to execute a FIND command that would retrieve values from more than one data set.
ROOT FILE CANNOT BE FOUND	This is a translation of IMAGE error 119. See Table B-1 for more information.

SAME LINES HAVE CONFLICTING REPORT OPTIONS	Like statements in a REPORT command (such as all DETAIL statements of same level) have conflicting SPACE or SKIP options. All items or literals that print on one line must not space or skip more or less than other statements that print on that line.
SELECT-FILE ERROR	An I/O error occurred while accessing the select-file.
SELECT-FILE NOT DECLARED	The user attempted to execute a FIND command before a select-file was specified. Enter the SELECT-FILE= command, then reenter the FIND command.
SELECT-FILE OVERFLOW	QUERY found more data entries than there is room for in the select file. A select file must be [illegible]
SERIAL READ IN PROGRESS	This is an indication to the user that QUERY is reading each record in the data set in order to satisfy the requirements of a FIND command.
SOF OR EOF READ OR SENSED	This is a translation of FMP error 12. See the RTE-IV Programmer's Reference Manual for more information.
SORT ERROR - XXXXXX	There was a numbered IMAGE error while reading entries from the database. XXXXXX represents the error code. See Table B-1 for an explanation of the code.
SORT LEVEL XX IS MISSING OR DUPLICATED	GROUP or TOTAL statements in a REPORT command reference sort levels that do not appear in a SORT statement.
SORT VALUE SIZE EXCEEDS LIMITS	Total SORT key size is longer than 80 characters.
STRING ADD ERROR	Error occurred in SADD routine during ASCII string ADD operation (string was [illegible]

SYNTAX ERROR	The command is not in the proper form. Retype the command.
TOTALLED STRING LENGTH EXCEEDS 20	An ASCII string (DECAR format) cannot exceed 20 characters if it is being added or averaged.
UNABLE TO SCHEDULE DBCOP	This is a translation of IMAGE error 140. See Table B-1 for more information.
UNOBTAINABLE OPEN MODE	This is a translation of IMAGE error 152. See Table B-1 for more information.
UNOBTAINABLE OPEN MODE	This is a translation of IMAGE error 152. See Table B-1 for more information.
UPDATE EXPECTED	A procedure file was used in an UPDATE command that did not include the UPDATE statement.
USER ACCESS NOT HIGH ENOUGH	The user did not have write access to the data set. Re-open the data base with a higher level of capability.
VALUE MUST HAVE QUOTES - ITEM IGNORED	A data item presented to QUERY must be enclosed in quotes.
VALUE TOO LARGE FOR PARAMETER	This is a translation of FMP error 30. See the RTE File Manager Manual for more information.
VALUE TOO LONG - ITEM IGNORED	The value entered for an UPDATE,ADD or REPLACE command is longer than the length defined for the item. Reenter the value.

QUERY, as a user of the IMAGE subroutines, can have a numbered error code returned to it. In that case, it will translate the numbered code to an error message and output this message to the user. The following is the translation table that it uses. Note that some of these messages may be difficult, if not impossible, to occur from QUERY.

Table B-7. IMAGE Error Codes and QUERY Messages

Numbered Code	Error Message
100	INVALID DATA SET NAME OR NUMBER
101	INVALID DATA ITEM NAME OR NUMBER
102	INVALID OR MISSING SEARCH OR SORT ITEM
103	DATABASE NOT OPENED
104	OPEN MODE DOES NOT ALLOW THAT OPERATION
105	DETAIL DATA SET IS FULL
106	MASTER DATA SET IS FULL
107	NO MASTER ENTRY WITH KEY ITEM VALUE
108	REQUEST DIRECTED AT AN AUTOMATIC MASTER
110	A MASTER ENTRY WITH KEY VALUE EXISTS
111	NO CURRENT CHAIN OR BAD RECORD NUMBER
112	CANNOT ALTER THE VALUE OF A KEY OR SORT ITEM
113	ENTRY BEING DELETED HAS NON-EMPTY CHAINS
114	RECORD IS EMPTY
115	INVALID MODE
116	FILE SPECIFIED IS NOT A ROOT FILE
117	ILLEGAL SECURITY CODE
118	DATA SET IS NOT WRITE ENABLED
119	ROOT FILE CANNOT BE FOUND
120	DATA SET FOR OPERATION MUST BE A DETAIL
121	DETAIL DATA SET HAS NO PATHS
122	CANNOT PERFORM A CHAIN READ

123	DATA SET FOR OPERATION MUST BE A MASTER
124	INVALID DBINF MODE
125	BAD DATA SET OR DATA ITEM IN DBINF CALL
128	PARTITION IS TOO SMALL
129	DATABASE OPEN EXCLUSIVELY
131	NO ROOM FOR DATABASE ENTRY IN DBCOP
132	NO RESOURCE NUMBER AVAILABLE
134	DATABASE NOT ENABLED FOR LOCKING
135	DATABASE CANNOT BE CLOSED DUE TO LOCK
136	DATABASE ALREADY LOCKED TO ANOTHER USER
137	DATABASE RN IS BEING USED ILLEGALLY
140	UNABLE TO SCHEDULE DBCOP
150	DATABASE ALREADY OPEN
152	UNOBTAINABLE OPEN MODE
153	DATABASE INACCESSIBLE WITH THAT LEVEL
154	DATABASE CORRUPT - BAD CHAIN POINTER
155	BEGINNING OR END OR CHAIN ENCOUNTERED
156	DETAIL CHAIN IS EMPTY
157	NO CURRENT RECORD FOR OPERATION
158	DATA ITEM IS NOT PART OF GIVEN DATA SET
159	DATABASE MUST BE LOCKED
160	INTERNAL BUFFERS ARE CORRUPT
161	INTERNAL DBCOP ERROR
162	INTERNAL IMAGE CALL IS INVALID

Appendix C
Converting 92063A To 92069A

This appendix contains information necessary to the users of the 92063A IMAGE/1000 who are converting to 92069A IMAGE. The enhancements in the product are highlighted, and the steps necessary for conversion are outlined. If additional information about an enhancement is necessary, the user is referred to the section in the manual that describes the feature.

It is recommended that conversion to RTE-IVB and 92069A IMAGE be in two steps. First the user should load his application programs on RTE-IVB using 92063A IMAGE, making sure that there are no problems in that conversion. Once the application programs are running on RTE-IVB with the old IMAGE, conversion to the new IMAGE can begin. Information on the loading of both IMAGE products in one RTE-IVB system is included at the end of this appendix.

IMAGE Enhancements

This section outlines the major enhancements added to IMAGE/1000 with the 92069A version. For more information about any feature, consult the page number listed in parenthesis.

Creating The Database

- Maximum data set capacity has been increased from 32,767 to (2**31)-1. The data set capacity is now limited by disc capacity rather than the number of entries per set since data sets cannot span disc volumes (2-6, 2-8).
- Maximum data entry size has been increased from 510 bytes to 4096 bytes (2-5).
- Maximum length of character items is increased from 126 to 255 characters (2-4).
- Data items can be compound items, with up to 255 elements in a compound item (2-4).

- The same data item names can now be used in different data sets (2-6, 2-9).
- The maximum number of paths in a master data set and key items in a detail data set have been increased from five to sixteen (2-6).
- The user has the option of having the Schema Processor purge the root file and data sets before re-creating them (2-13).
- Key items no longer require a write level of 15 (2-3).
- Database and data set names can have up to six characters rather than five (iii).

Querying The Database

- QUERY can now be run in batch mode as well as interactively (3-3).
- A new command, XEQ, has been added that allows the execution of a command file while executing QUERY interactively (3-48).
- QUERY can be executed at a remote DS/1000 node for remote data base access (3-3).
- Real numbers can be reported (similar to a FORTRAN G13.5 format) in a QUERY report (3-30).
- Report detail entries can now take up to ten lines rather than one (3-27).
- Reports can be written to a file as well as a device (3-52).
- QUERY will create a select file if the file named does not exist (3-6).
- QUERY select files will include a header. Information in this header will facilitate use of the select file outside of QUERY (7-16).
- All error messages will be fully reported in ASCII error messages rather than just numbered errors (B-6).
- The alphanumeric edit mask size has been increased from 72 to 132 characters (3-28).

Host Language Access

- The parameter strings for the IMAGE subroutines have been revised so as to be consistent from call to call. With a few exceptions, the first four parameters in each subroutine call will be the same (4-16).
- The subroutine DBINT and the requirement to use it have been eliminated (4-1).
- Remote database access is easily accomplished by passing the node number as part of the database namr string (4-17).
- With FORTRAN or Assembly Language, data sets and data items can be accessed by name as well as number (4-18, 4-19).
- All modifications to the database are posted immediately regardless of open mode (C-15).
- Three additional DBGET modes have been added: reread current record, backward serial read and backward chain read (4-29, 4-33).
- The modes and return information for DBINF have been changed (4-37 through 4-41).
- More than one database can be opened to a program (4-1).
- If the database is opened in shared read/write mode (mode 1), the database must be locked before an update can be done (4-14).

Maintaining The Database

- The backup utilities, DBLOD, DBULD, DBSTR and DBRST, can now be used in batch mode. All parameters can be passed in the RU string (6-3, 6-7, 6-9, 6-11).
- The backup utilities can back up the database to a device or a file without going through the Batch Spool Monitor (6-3, 6-7, 6-9, 6-11).

Miscellaneous Enhancements

- FMP file namrs will be used whenever a file is referenced, such as database name, QUERY select file, etc. (iii).
- A more efficient hashing algorithm is used to reduce the number of synonyms (7-13).

Conversion Steps

The user wishing to convert from 92063A IMAGE to the new IMAGE should follow the following steps.

1. Unload the old database using the DBUP utility.
2. Modify the schema to conform to the new Schema Processor requirements.
3. Use the new Schema Processor to create a new root file and data sets.
4. Reload the new database with the old data, using the new DBLOD utility.
5. Modify any existing programs to reflect the changes in calling sequences and elimination of DBINT.

DBUP

DBUP is a utility designed to unload a 92063A IMAGE database, converting it to a format which can be reloaded into a 92069A database. It is the only method available for transporting databases from the old to the new IMAGE.

DBUP should be loaded using the 92063A IMAGE routines. It should be loaded in the same way that other IMAGE programs were loaded which includes allowing additional space for the Run Table and specifying OP,SS to allow for use of SSGA.

If the storage medium is to be a magnetic tape, it is necessary to create a type 0 file called MT, with write capability. The following File Manager Command is necessary.

```
     CR,MT::crn:0,tplu,WR
```

where:

crn	is a cartridge reference number to which the user has write access
tplu	is the magnetic tape lu.

The type 0 file must be the first file encountered which has the name MT since DBUP will not have access to the crn.

If the storage device is to be a disc file, no special action is required.

Running DBUP

The following command is used to schedule DBUP.

```
:RU,DBUP,console,storage,root,level,abort
```

where:

console	is the console logical unit (lu). Default is the scheduling lu.
storage	is the FMP namr where the data is to be stored. Default is lu 8. The device cannot be a mini-cartridge unit.
root	is the FMP file namr of the database root file. If omitted, DBUP asks the user interactively for the file namr on the console lu.
level	is the highest level word defined for the database or any word if no level words were defined in the database schema. If omitted, DBUP asks the user interactively for a level word on the console lu.
abort	is AB to abort DBUP if the end of the storage device is reached; CO if the user wishes to continue under this condition. If omitted, DBUP asks the user interactively for the abort word on the console lu.

For an example, see the description of DBULD in Chapter 6. The calling sequence is exactly the same as for DBUP.

Loading the Database

Once the database has been unloaded using DBUP, it can easily be reloaded for use with new IMAGE. This should be done using the DBLOD utility which has been loaded using the new IMAGE. Information about how to use DBLOD is found in Chapter 6. A file created by DBUP cannot be transferred via a FMGR ST command. Records will be truncated and the data cannot be reloaded.

Converting The Schema

Certain changes must be made to a schema in order to use it with the new IMAGE. The changes to be made are discussed in this section. The changes fall into one of two categories - mandatory and optional. Following the discussion of these changes two schemas are shown - one from 92063A and the same schema converted to 92069A or 92073A. Certain lines are marked with numbers corresponding to the numbers on the descriptions of the changes.

The mandatory changes are:

1. A colon (:) must be inserted between the word CONTROL and the control options.

2. A colon (:) must be inserted after "BEGIN DATABASE".

3. The database name, security code and cartridge in the BEGIN DATA BASE statement are now passed as an FMP file name.

4. All U type item descriptions (character type) must be changed to X.

5. The set name entries should be changed from SETNAM, TYPE, CRXX to SETNAMR,TYPE.

Some important optional changes are:

6. The key items in detail data sets can use the same definitions as the related key items in master sets.

7. Key items can have their write levels reduced from 15.

8. Repetitive data items can be put in a compound item.

9. The number of keys in a detail or paths in a master can be increased to more than five.

The following schema is from the 92063A IMAGE.

```
$CONTROL;
BEGIN DATABASE QA;CR043;100;
LEVELS:
       1     OPER;
       5     ACCT;
      10     ADMIN;
      15     SECUR;
ITEMS:
       ASSY#,  U10(1,15);  << ASSEMBLY NUMBER FOR ASSEMBLY MASTER >>
                           << AND FAILURE FILE.                   >>
       ASSY1,  U10(1,15);
       FAIL#,  I1(1,10);   << NUMBER OF FAILURES - ASSEMBLY       >>
       PART#,  U8(1,15);   << PART NUMBER FOR PART NUMBER MASTER  >>
                           << AND FAILURE FILE                    >>
       PART1,  U8(1,15);
       FAILCD, U4(1,15);   << FAIL CODE                           >>
       FAILC1, U4(1,15);
       REFDES, U4(1,15);   << REFERENCE DESIGNATOR                >>
       REFDS1, U4(1,15);
       DATE,   U6(1,15);   << DATE FOR FAILURE FILE, DATE MASTER  >>
                           << AND PRODUCT TESTED FILE             >>
       DAT1,   U6(1,15);
       DAT2,   U6(1,15);
       EMP#,   U6(1,15);   << EMPLOYEE NUMBER FOR EMPLOYEE MASTER >>
                           << FAILURE FILE AND TIME VOUCHER FILE  >>
       EMP1,   U6(1,15);
       EMP2,   U6(1,15);
       PROJ#,  U6(1,15);   << PROJECT NUMBER FOR PROJECT NUMBER   >>
                           << MASTER AND TIME VOUCHER FILE        >>
       PROJ1,  U6(1,15);
       DEPT#,  U4(1,5);    << DEPARTMENT NUMBER                   >>
       OPDAT,  U6(1,5);    << OPEN DATE FOR PROJECT NUMBER        >>
       CLDAT,  U6(1,5);    << CLOSE DATE FOR PROJECT NUMBER       >>
       PNT1,   U2(1,1);    << FAIL/NO FAIL ARRAY FOR EACH PIN     >>
       PNT2,   U2(1,1);
       PNT3,   U2(1,1);
       PNT4,   U2(1,1);
       PNT5,   U2(1,1);
       PNT6,   U2(1,1);
       PNT7,   U2(1,1);
       PNT8,   U2(1,1);
       HOURS,  R2(1,1);    << HOURS PER PROJECT NUMBER            >>
SETS:
<<                                                                >>
<<                ASSEMBLY FILE                                   >>
<<                                                                >>
NAME:  ASMBY,M,CR043;      << ASSEMBLY FILE CONTAINS THE ASSEMBLY >>
                           << NUMBER AND TOTAL NUMBER OF FAILURES >>
                           << FOR THAT ASSEMBLY                   >>
```

```
ENTRY: ASSY#(1),
       FAIL#;
CAPACITY:409;
<<                                                                   >>
<<                PART NUMBER FILE                                   >>
<<                                                                   >>
NAME:  PART,M,CR043;        << PART FILE CONTAINS ONLY PART NUMBER.>>
                            << NOTE THAT ALTHOUGH IT CONTAINS ONLY >>
                            << ONE ITEM, IT IS A MANUAL MASTER SO  >>
                            << ENTRIES CAN BE CONTROLLED.          >>
ENTRY: PART#(1);
CAPACITY: 4001;
<<                                                                   >>
<<                FAIL CODE FILE                                     >>
<<                                                                   >>
NAME:  CODE,A,CR043;        << FAIL CODE FILE CONTAINS ALL FAILURE >>
                            << CODES                               >>
ENTRY: FAILCD(1);
CAPACITY: 997;
<<                                                                   >>
<<                DATE FILE                                          >>
<<                                                                   >>
NAME:  DATEF,A,CR043;       << DATE FILE ALLOWS ACCESS BY GIVEN DAY>>
ENTRY: DATE(2);
CAPACITY: 367;
<<                                                                   >>
<<                REFERENCE DESIGNATOR FILE                          >>
<<                                                                   >>
NAME:  REF,A,CR043;         << REFERENCE DESIGNATOR FILE ALLOWS    >>
                            << ACCESS BY DESIGNATOR                >>
ENTRY: REFDES(1);
CAPACITY: 997;
<<                                                                   >>
<<                EMPLOYEE FILE                                      >>
<<                                                                   >>
NAME:  EMPL,A,CR043;        << EMPLOYEE FILE ALLOWS ACCESS BY EMP# >>
                            << TO FAILURE FILE AND TIME VOUCHER    >>
                            << FILE.                               >>
ENTRY: EMP#(1);
CAPACITY: 263;
<<                                                                   >>
<<                PROJECT NUMBER FILE                                >>
<<                                                                   >>
NAME:  PROJ,M,CR043;        << PROJECT NUMBER FILE HAS ASSOCIATED  >>
                            << DEPARTMENT NUMBER AND OPENING AND   >>
                            << CLOSING DATES OF NUMBER.            >>
ENTRY: PROJ#(1),
       DEPT#,
       OPDAT,
       CLDAT;
CAPACITY: 101;
```

```
<<                                                                >>
<<                 FAILURE FILE                                   >>
<<                                                                >>
NAME:  FAIL,D,CR043;       << FAILURE FILE HAS A TRANSACTION      >>
                           << ENTERED WHENEVER THE TEST TECHNICIAN>>
                           << DETECTS A FAILURE                   >>
ENTRY: ASSY1(ASMBY),
       DAT1(DATEF),
       PART1(PART),
       FAILC1(CODE),
       REFDS1(REF),
       PNT1,
       PNT2,
       PNT3,
       PNT4,
       PNT5,
       PNT6,
       PNT7,
       PNT8,
       EMP1;
CAPACITY: 1009;
<<                                                                >>
<<                 TIME VOUCHER FILE                              >>
<<                                                                >>
NAME:  TIME,D,CR043;       << TIME VOUCHER FILE CONTAINS EMPLOYEE >>
                           << NUMBER, DEPARTMENT NUMBER, DATE AND >>
                           << HOURS.                              >>
ENTRY: EMP2(EMPL),
       PROJ1(PROJ),
       DAT2(DATEF),
       HOURS;
CAPACITY: 367;
END.
```

The following schema is from the 92069A IMAGE.

```
$CONTROL: TABLE;
BEGIN DATABASE: QA:100:43;
LEVELS:
       1    OPER;
       5    ACCT;
      10    ADMIN;
      15    SECUR;
ITEMS:
       ASSY#,  X10(1,10);   << ASSEMBLY NUMBER FOR ASSEMBLY MASTER >>
                            << AND FAILURE FILE.                   >>
       FAIL#,  I1(1,10);    << NUMBER OF FAILURES - ASSEMBLY       >>
       PART#,  X8(1,10);    << PART NUMBER FOR PART NUMBER MASTER  >>
                            << AND FAILURE FILE                    >>
       FAILCD, X4(1,10);    << FAIL CODE                           >>
       REFDES, X4(1,10);    << REFERENCE DESIGNATOR                >>
       DATE,   X6(1,1);     << DATE FOR FAILURE FILE, DATE MASTER  >>
                            << AND PRODUCT TESTED FILE             >>
       EMP#,   X6(1,10);    << EMPLOYEE NUMBER FOR EMPLOYEE MASTER >>
                            << FAILURE FILE AND TIME VOUCHER FILE  >>
       PROJ#,  X6(1,5);     << PROJECT NUMBER FOR PROJECT NUMBER   >>
                            << MASTER AND TIME VOUCHER FILE        >>
       DEPT#,  X4(1,5);     << DEPARTMENT NUMBER                   >>
       OPDAT,  X6(1,5);     << OPEN DATE FOR PROJECT NUMBER        >>
       CLDAT,  X6(1,5);     << CLOSE DATE FOR PROJECT NUMBER       >>
       PNTS,8  X1(1,1);     << FAIL/NO FAIL ARRAY FOR EACH PIN     >>
       HOURS,  R2(1,1);     << HOURS PER PROJECT NUMBER            >>
SETS:
<<                                                                 >>
<<                ASSEMBLY FILE                                    >>
<<                                                                 >>
NAME:  ASMBY::43,M;         << ASSEMBLY FILE CONTAINS THE ASSEMBLY >>
                            << NUMBER AND TOTAL NUMBER OF FAILURES >>
                            << FOR THAT ASSEMBLY                   >>
ENTRY: ASSY#(1),
       FAIL#;
CAPACITY:409;
<<                                                                 >>
<<                PART NUMBER FILE                                 >>
<<                                                                 >>
NAME:  PART::43,M;          << PART FILE CONTAINS ONLY PART NUMBER.>>
                            << NOTE THAT ALTHOUGH IT CONTAINS ONLY >>
                            << ONE ITEM, IT IS A MANUAL MASTER SO  >>
                            << ENTRIES CAN BE CONTROLLED.          >>
ENTRY: PART#(1);
CAPACITY: 4001;
<<                                                                 >>
<<                FAIL CODE FILE                                   >>
<<                                                                 >>
NAME:  CODE::43,A;          << FAIL CODE FILE CONTAINS ALL FAILURE >>
```

```
                          << CODES                                 >>
ENTRY: FAILCD(1);
CAPACITY: 997;
<<                                                                 >>
<<                 DATE FILE                                       >>
<<                                                                 >>
NAME:  DATEF::43,A;       << DATE FILE ALLOWS ACCESS BY GIVEN DAY>>
ENTRY: DATE(2);
CAPACITY: 367;
<<                                                                 >>
<<                 REFERENCE DESIGNATOR FILE                       >>
<<                                                                 >>
NAME:  REF::43,A;         << REFERENCE DESIGNATOR FILE ALLOWS      >>
                          << ACCESS BY DESIGNATOR                  >>
ENTRY: REFDES(1);
CAPACITY: 997;
<<                                                                 >>
<<                 EMPLOYEE FILE                                   >>
<<                                                                 >>
NAME:  EMPL::43,A;        << EMPLOYEE FILE ALLOWS ACCESS BY EMP#  >>
                          << TO FAILURE FILE AND TIME VOUCHER     >>
                          << FILE.                                >>
ENTRY: EMP#(2);
CAPACITY: 263;
<<                                                                 >>
<<                 PROJECT NUMBER FILE                             >>
<<                                                                 >>
NAME:  PROJ::43,M;        << PROJECT NUMBER FILE HAS ASSOCIATED   >>
                          << DEPARTMENT NUMBER AND OPENING AND    >>
                          << CLOSING DATES OF NUMBER.             >>
ENTRY: PROJ#(1),
       DEPT#,
       OPDAT,
       CLDAT;
CAPACITY: 101;
<<                                                                 >>
<<                 FAILURE FILE                                    >>
<<                                                                 >>
NAME:  FAIL::43,D;        << FAILURE FILE HAS A TRANSACTION       >>
                          << ENTERED WHENEVER THE TEST TECHNICIAN>>
                          << DETECTS A FAILURE                    >>
ENTRY: ASSY#(ASMBY),
       DATE(DATEF),
       PART#(PART),
       FAILCD(CODE),
       REFDES(REF),
       PNTS,
       EMP#(EMPL);
CAPACITY: 1009;
<<                                                                 >>
<<                  TIME VOUCHER FILE                              >>
<<                                                                 >>
NAME:  TIME::43,D;        << TIME VOUCHER FILE CONTAINS EMPLOYEE >>
                          << NUMBER, DEPARTMENT NUMBER, DATE AND >>
                          << HOURS.                              >>
ENTRY: EMP#(EMPL),
       PROJ#(PROJ),
       DATE(DATEF),
       HOURS;
CAPACITY: 1000;
END.
```

Converting Application Programs

This section discusses the conversion of FORTRAN application programs. BASIC and Assembly Language programmers will find the parameter differences and the information in the miscellaneous notes pertain at least in part to all languages. For each subroutine, there is a translation to the new format, plus miscellaneous notes regarding use of the subroutine.

Initializing a Database (DBINT)

Remove any DBINT calls.

Opening a Database (DBOPN)

```
     CALL DBOPN(IBASE,ILEVL,ISCOD,IMODE,ISTAT)
```

New format:

```
     CALL DBOPN(IBASE,ILEVL,IMODE,ISTAT)
```

The IBASE parameter is now a string containing an FMP file namr, with an integer node number or two blanks in the first word. Open modes are now 1, 3 and 8 rather than 1, 2 and 3. ISCOD is now included in the IBASE parameter as part of the FMP fill namr. Table C-1 shows the correspondence.

TABLE C-1. Old and New Open Modes

OLD	NEW
1	8
2	1
3	3

The correspondence is not exact because the old mode 1 allowed sharing with mode 2 open. The new mode 8 does not allow simultaneous open with mode 1 users, but only with other mode 8 opens. The same is true for new mode 1. Sharing is only with other mode 1 users.

Database Information (DBINF)

Old format:

```
     CALL DBINF(ITYPE,IMODE,ID,IBUF)
```

New format:

```
     CALL DBINF(IBASE,ID,IMODE,ISTAT,IBUF)
```

The IBASE parameter is the same array referenced in the DBOPN call. The mode parameters and the information returned have been completly changed. The user should read the information on DBINF in Chapter 4 to determine what changes should be made. The user should also reevaluate the use of DBINF. Some information which was necessary, for instance, a data item number when only a name was available, is no longer needed.

One thing to note is that the modes of DBINF that were previously referred to as save and restore run table are now modes 401 and 402, save and restore chain information.

Reading the Database (DBGET)

Old format:

```
     CALL DBGET(IDSET,IMODE,ISTAT,IBUF,IARG)
```

New format:

```
     CALL DBGET(IBASE,ID,IMODE,ISTAT,LIST,IBUF,IARG)
```

IBASE is the same array referenced in the DBOPN call. When doing a chain read, a status code of 155 is now returned in ISTAT(1) to indicate that the end of the chain has been reached. This eliminates the need to check ISTAT(4).

When doing a serial read, the end of the file is indicated by a status code of 12 in ISTAT(1). This is in lieu of the zero that is currently found in ISTAT(2).

In BASIC, the NAME-LIST(LIST) parameter is reduced to 9 items maximum from 10.

Updating a Database (DBUPD)

Old format:

```
     CALL DBUPD(IDSET,ISTAT,INBR,IVALU,IBUF)
```

New format:

```
     CALL DBUPD(IBASE,IDSET,IMODE,ISTAT,INBR,IVALU)
```

IBASE is the same array referenced in the DBOPN call. IMODE must be an integer of value 1. IBUF is not used.

If the database is opened in shared read/write mode (mode 1), the database must be locked before an update can be performed on it.

In BASIC, the NAME-LIST(LIST) parameter is reduced to 10 item names maximum from 12.

Database Write (DBPUT)

Old format:

```
     CALL DBPUT(IDSET,ISTAT,INBR,IVALU,IBUF)
```

New format:

```
     CALL DBPUT(IBASE,IDSET,IMODE,ISTAT,INBR,IVALU)
```

IBASE is the same array referenced in the DBOPN call. IMODE must be an integer of value 1. IBUF is not used.

In BASIC, the NAME-LIST(LIST) parameter is reduced to 10 item names maximum from 12.

Locate Master Data Entry (DBFND)

Old format:

```
     CALL DBFND(ISTAT,IDSET,IPATH,IARG)
```

New format:

```
     CALL DBFND(IBASE,IDSET,IMODE,ISTAT,IPATH,IARG)
```

IBASE is the same array referenced in the DBOPN call. IMODE must be an integer of value 1. If the key item desired exists in the master data set, but no entries are in the chain, a status code of 156 is returned in ISTAT(1). This eliminates the need to check ISTAT(3) for the chain length.

Deleting a Record (DBDEL)

Old format:

```
     CALL DBDEL(IDSET,ISTAT)
```

New format:

```
     CALL DBDEL(IBASE,IDSET,IMODE,ISTAT)
```

IBASE is the same array referenced in the DBOPN call. IMODE must be an integer of value 1.

Close the Database (DBCLS)

Old format:

```
     CALL DBCLS(IMODE,ISTAT)
```

New format:

```
     CALL DBCLS(IBASE,ID,IMODE,ISTAT)
```

IBASE is the same array referenced in the DBOPN call. The possible values for IMODE are 1 (close a database), or 2 (close a data set). If the value of IMODE is 1, ID is ignored. The new IMODE = 1 is equivalent to the old IMODE = 0.

There is no equivalent to the old IMODE = 1 (post the root file). Post mode close is no longer necessary since IMAGE automatically posts the database to disc after every modification.

Database Lock (DBLCK)

Old format:

```
     CALL DBLCK(IMODE,ISTAT)
```

New Format:

```
     CALL DBLCK(IBASE,ID,IMODE,ISTAT)
```

IBASE is the same array referenced in the DBOPN call. ID is a dummy parameter. Formerly the status code 104 was returned if the user was not in the proper open mode to lock. This has been changed to 134.

Database Unlocking (DBUNL)

Old format:

```
     CALL DBUNL(ISTAT)
```

New format

```
     CALL DBUNL(IBASE,ID,IMODE,ISTAT)
```

IBASE is the same array referenced in the DBOPN call. ID is a dummy parameter. IMODE must be an integer of value 1.

Loading Both IMAGE Products Into One System

As conversion takes place from 92063A to 92069A IMAGE it will be useful to have both products up on RTE-IVB system at the same time. There are a few considerations the user must keep in mind in order to achieve this.

The procedure used is to generate one product in and load the second product on-line. It is recommended that the library from the 92063A product be generated in because of the need for .DBRN to be in SSGA. DBCOP from 92069A can also be generated in without being affected by the old product. If the 92063A product is generated in, the Remote Monitors, RDBAM and RDBAP cannot be. This is not a serious problem, since they can be loaded on-line. Only RD.TB from 92069A must be generated in (in SSGA) and it does not conflict with the old library.

Once the generation is complete, programs such as QUERY, DBDS and the utilities need to be loaded on-line. The procedure is to load one or the other version of a program, save it via the SP command and rename the main only. Do not rename any segments. Once the first program is loaded, saved and renamed, its counterpart in the other product can be loaded and saved.

Precautions should be taken such that when programs are loaded, the proper combination of code and library exist. For example, programs using the old IMAGE formats should be loaded using the 92063A library.

Appendix D
92069A vs. 92073A Image

Differences between 92069A and 92073A IMAGE/1000 stem from architectural and operating system differences between the HP 1000 M/E/F and L-series computers. 92073A IMAGE/1000 contains only those IMAGE features consistent with the application and capability of the L-series. Not found are QUERY and access from BASIC. Additionally, database capacity and remote database access differ slightly between operating systems.

The 92073A IMAGE/1000 product is also suitable for RTE-IVB users not desiring QUERY and access from BASIC. If QUERY is not required, users may choose the smaller product for its economy and simplicity.

QUERY

QUERY is not found in 92073A IMAGE because of the absence of system tracks in RTE-L. QUERY uses system tracks for sorting of entries before they are REPORTed. Without system tracks, QUERY cannot be included in 92073A IMAGE/1000.

BASIC

IMAGE may not be accessed from BASIC under RTE-L because of the way the BASIC/IMAGE interface is implemented. It uses a separate overlay program to resolve data representation incompatibilities between the two subsystems. Since RTE-L has only one background partition, the user's BASIC interpreter and the overlay program would be forced to swap. This swapping would degrade performance beyond reason. Therefore, the BASIC/IMAGE interface is not found in 92073A IMAGE/1000.

CAPACITY

The File Management Package (FMP) of RTE-L or of RTE-XL uses one computer word to represent addresses in a file. For IMAGE, this single-word addressing means that data set capacity on RTE-L is limited to 32767 entries. In contrast, RTE-IVB uses two computer words to address files (double-word addressing). For this reason, data sets on RTE-IVB may have as many as (2**31)-1 (over 2 billion) entries. This lets the larger computer support larger databases while the smaller computer saves time and space by only using only one word for addresses.

RDBA

The interface between IMAGE and DS/1000 is also dependent on the underlying operating system. Specifically, databases on RTE-L nodes cannot be accessed remotely. The more usual situation will feature a central RTE-IVB database being accessed by a number of satellite RTE-L nodes.

Two programs must be present on a database's node to enable remote access: RDBAM and RDBAP. RDBAM may not be run on RTE-L because it uses an executive request not supported there. Without RDBAM present, remote access of databases on RTE-L nodes is not possible.

Given the capabilities and expected applications of the L-series computer, these limitations are not seen as severe. If QUERY and access from BASIC are a must, perhaps the additional power of an M, E, or F-Series is required.

Index

READER COMMENT SHEET

92069A and 92073A IMAGE/1000
DATA BASE MANAGEMENT SYSTEM
Reference Manual

92069-90001 **July 1980**

Update No. __________
(If Applicable)

We welcome your evaluation of this manual. Your comments and suggestions help us improve our publications. Please use additional pages if necessary.

FROM:

Name	______________________________
Company	______________________________
Address	______________________________
	______________________________
	______________________________

FOLD FOLD

NO POSTAGE
NECESSARY
IF MAILED
IN THE
UNITED STATES

BUSINESS REPLY MAIL

FIRST CLASS PERMIT NO. 141 CUPERTINO, CA.

— POSTAGE WILL BE PAID BY —

Hewlett-Packard Company
Data Systems Division
11000 Wolfe Road
Cupertino, California 95014
ATTN: Technical Marketing Dept.

FOLD FOLD

SALES & SUPPORT OFFICES

Arranged alphabetically by country

Product Line Sales/Support Key

Key	Product Line
A	Analytical
CM	Components
C	Computer Systems
CP	Computer Systems Primary Service Responsible Office (SRO)
CS	Computer Systems Secondary SRO
E	Electronic Instruments & Measurement Systems
M	Medical Products
MP	Medical Products Primary SRO
MS	Medical Products Secondary SRO
P	Consumer Calculators

* Sales only for specific product line
** Support only for specific product line

IMPORTANT: These symbols designate general product line capability. They do not insure sales or support availability for all products within a line, at all locations. Contact your local sales office for information regarding locations where HP support is available for specific products.

HP distributors are printed in italics.

ANGOLA

Telectra
Empresa Técnica de Equipamentos Eléctricos, S.A.R.L.
R. Barbosa Rodrigues, 41-I DT.
Caixa Postal 6487
LUANDA
Tel: 35515,35516
A,E,M,P*

ARGENTINA

Hewlett-Packard Argentina S.A.
Avenida Santa Fe 2035
Martinez 1640 **BUENOS AIRES**
Tel: 798-5735, 792-1293
Telex: 122443 AR CIGY
Cable: HEWPACKARG
A,E,CP,P

Biotron S.A.C.I.y.M
Avenida Paseo Colon 221
9 Piso
1399 ***BUENOS AIRES***
Tel: 30-4846, 30-1851, 30-8384
Telex: (33)17595 BIONAR
Cable: BIOTRON Argentina
M

Fate S.A. Electronica
Bartolomeu Mitre 833
1036 ***BUENOS AIRES***
Tel: 74-41011, 74-49277, 74-43459
Telex: 18137, 22754
P

AUSTRALIA

Adelaide, South Australia Pty. Ltd.

Hewlett-Packard Australia Pty.Ltd.
153 Greenhill Road
PARKSIDE, S.A. 5063
Tel: 272-5911
Telex: 82536
Cable: HEWPARD Adelaide
A*,CM,CS,E,MS,P

Brisbane, Queensland Office

Hewlett-Packard Australia Pty.Ltd.
5th Floor
Teachers Union Building
495-499 Boundary Street
SPRING HILL, Queensland 4000
Tel: 229-1544
Telex: 42133
Cable: HEWPARD Brisbane
A,CM,CS,E,MS,P

Canberra, Australia Capital Office

Hewlett-Packard Australia Pty.Ltd.
121 Wollongong Street
FYSHWICK, A.C.T. 2609
Tel: 952733
Telex: 62650
Cable: HEWPARD Canberra
A*,CM,CS,E,MS,P

Melbourne, Victoria Office

Hewlett-Packard Australia Pty.Ltd.
31-41 Joseph Street
BLACKBURN, Victoria 3130
Tel: 89-6351
Telex: 31-024
Cable: HEWPARD Melbourne
A,CM,CP,E,MS,P

Perth, Western Australia Office

Hewlett-Packard Austalia Pty.Ltd.
141 Stirling Highway
NEDLANDS, W.A. 6009
Tel: 386-5455
Telex: 93859
Cable: HEWPARD Perth
A,CM,CS,E,MS,P

Sydney, New South Wales Office

Hewlett-Packard Australia Pty.Ltd.
17-23 Talavera Road
NORTH RYDE, N.S.W. 2113
P.O. Box 308
Tel: 887-1611
Telex: 21561
Cable: HEWPARD Sydney
A,CM,CP,E,MS,P

AUSTRIA

Hewlett-Packard Ges.m.b.h.
Grottenhofstrasse 94
Verkaufsburo Graz
8052 **GRAZ**
Tel: 21-5-66
Telex: 32375
CM,C*,E*

Hewlett-Packard Ges.m.b.h.
Wehlistrasse 29
P.O. Box 7
A-1205 **VIENNA**
Tel: (222) 35-16-210
Telex: 135823/135066
A,CM,CP,E,MS,P

BAHRAIN

Green Salon
P.O. Box 557
BAHRAIN
Tel: 5503
Telex: 88419
P

Wael Pharmacy
P.O. Box 648
BAHRAIN
Tel: 54886, 56123
Telex: 8550 WAEL GJ
M

BELGIUM

Hewlett-Packard Belgium S.A./N.V.
Blvd de la Woluwe, 100
Woluwedal
B-1200 ***BRUSSELS***
Tel: (02) 762-32-00
Telex: 23-494 paloben bru
A,CM,CP,E,MP,P

BRAZIL

Hewlett-Packard do Brasil I.e.C. Ltda.
Alameda Rio Negro, 750
ALPHAVILLE 06400 Barueri SP
Tel: 421-1311
Telex: 011 23602 HPBR-BR
Cable: HEWPACK Sao Paulo
A,CM,CP,E,MS

Hewlett-Packard do Brasil I.e.C. Ltda.
Rua Padre Chagas, 32
90000-**PORTO ALEGRE**-RS
Tel: 22-2998, 22-5621
Cable: HEWPACK Porto Alegre
A*,CM,E,MS,P*

Hewlett-Packard do Brasil I.e.C. Ltda.
Avenida Epitacio Pessoa, 4664
22471 **RIO DE JANEIRO**-RJ
Tel: 286-0237
Telex: 021-21905 HPBR-BR
Cable: HEWPACK Rio de Janeiro
A,CM,E,MS,P*

BURUNDI

Typomeca S.P.R.L.
B.P. 553
BUJUMBURA
Tel: 2659
P

CANADA

Alberta

Hewlett-Packard (Canada) Ltd.
210, 7220 Fisher Street S.E.
CALGARY, Alberta T2H 2H8
Tel: (403) 253-2713
Telex: 610-821-6141
A,CM,CP,E*,MS,P*

Hewlett-Packard (Canada) Ltd.
11620A-168th Street
EDMONTON, Alberta T5M 3T9
Tel: (403) 452-3670
Telex: 610-831-2431
A,CM,CP,E,MS,P*

British Columbia

Hewlett-Packard (Canada) Ltd.
10691 Shellbridge Way
RICHMOND, British Columbia V6X 2W7
Tel: (604) 270-2277
Telex: 610-922-5059
A,CM,CP,E*,MS,P*

Manitoba

Hewlett-Packard (Canada) Ltd.
380-550 Century Street
WINNIPEG, Manitoba R3H 0Y1
Tel: (204) 786-7581
Telex: 610-671-3531
A,CM,CS,E,MS,P*

Nova Scotia

Hewlett-Packard (Canada) Ltd.
P.O. Box 931
900 Windmill Road
DARTMOUTH, Nova Scotia B2Y 3Z6
Tel: (902) 469-7820
Telex: 610-271-4482
CM,CP,E*,MS,P*

Ontario

Hewlett-Packard (Canada) Ltd.
552 Newbold Street
LONDON, Ontario N6E 2S5
Tel: (519) 686-9181
Telex: 610-352-1201
A,CM,CS,E*,MS,P*

Hewlett-Packard (Canada) Ltd.
6877 Goreway Drive
MISSISSAUGA, Ontario L4V 1M8
Tel: (416) 678-9430
Telex: 610-492-4246
A,CM,CP,E,MP,P

Hewlett-Packard (Canada) Ltd.
1020 Morrison Drive
OTTAWA, Ontario K2H 8K7
Tel: (613) 820-6483
Telex: 610-563-1636
A,CM,CP,E*,MS,P*

Quebec

Hewlett-Packard (Canada) Ltd.
17500 South Service Road
Trans-Canada Highway
KIRKLAND, Quebec H9J 2M5
Tel: (514) 697-4232
Telex: 610-422-3022
A,CM,CP,E,MP,P*

CHILE

Jorge Calcagni y Cia. Ltda.
Arturo Burhle 065
Casilla 16475
Correo 9, ***SANTIAGO***
Tel: 220222
Telex: JCALCAGNI
A,E,M,P

Olympia (Chile) Ltd.
Rodrico de Araya 1045
Casilla 256-V
SANTIAGO *21*
Tel: 25-50-44
Telex: 40-565
P

COLOMBIA

Instrumentación
H. A. Langebaek & Kier S.A.
Apartado Aéreo 6287
BOGOTÁ *1, D.E.*
Carrera 7 No. 48-75
BOGOTA, *2 D.E.*
Tel: 287-8877
Telex: 44400
Cable: AARIS Bogota
A,E,M,P

Instrumentación
H. A. Langebaek & Kier S.A.
Edif. Camacol, Local 105
Carrera 63 NO. 49-A-31
Apartado 54098
MEDELLIN
Tel: 304475
A,E,M,P

COSTA RICA

Cientifica Costarricense S.A.
Avenida 2, Calle 5
San Pedro de Montes de Oca
Apartado 10159
SAN JOSÉ
Tel: 24-38-20, 24-08-19
Telex: 2367 GALGUR
Cable: GALGUR
A,E,M

CYPRUS

Telerexa Ltd.
P.O. Box 4809
14C Stassinos Avenue
NICOSIA
Tel: 45628
Telex: 2894
E,M,P

CZECHOSLOVAKIA

Hewlett-Packard
Obchodni Zastupitelstvi v CSSR
Pisemny Styk
Post. schranka 27
CS-118 01 ***PRAHA 011***
Tel: 66-296
Telex: 121353 IHC

DENMARK

Hewlett-Packard A/S
Datavej 52
DK-3460 **BIRKEROD**
Tel: (02) 81-66-40
Telex: 37409 hpas dk
A,CM,CP,E,MS,P

Hewlett-Packard A/S
Navervej 1
DK-8600 **SILKEBORG**
Tel: (06) 82-71-66
Telex: 37409 hpas dk
CM,CS,E

ECUADOR

CYEDE Cia. Ltda.
P.O. Box 6423 CCI
Avenida Eloy Alfaro 1749
QUITO
Tel: 450-975, 243-052
Telex: 2548 CYEDE ED
Cable: CYEDE-Quito
A,E

Hospitalar S.A.
Casilla 3590
Robles 625
QUITO
Tel: 545-250, 545-122
Cable: HOSPITALAR-Quito
M

EGYPT

Samitro
Sami Amin Trading Office
18 Abdel Aziz Gawish
ABDINE-CAIRO
Tel: 24-932
P

International Engineering Associates
24 Hussein Hegazi Street
Kasr-el-Aini
CAIRO
Tel: 23-829
Telex: 93830
E,M

Informatic For Computer Systems
22 Talaat Harb Street
CAIRO
Tel: 759006
Telex: 93938 FRANK UN
C

SALES & SUPPORT OFFICES
Arranged alphabetically by country

EL SALVADOR
IPESA
Boulevard de los Heroes
Edificio Sarah 1148
SAN SALVADOR
Tel: 252787
A,C,E,P*

FINLAND
Hewlett-Packard Oy
Revontulentie 7
SF-02100 **ESPOO** 10
Tel: (90) 455-0211
Telex: 121563 hewpa sf
A,CM,CP,E,MS,P

FRANCE
Hewlett-Packard France
Le Ligoures
Bureau de Vente de
Aix-en-Provence
Place Romée de Villeneuve
F-13090 **AIX-EN-PROVENCE**
Tel: (42) 59-41-02
Telex: 410770F
A,CM,CS,E,MS,P*

Hewlett-Packard France
Boite Postale No. 503
F-25026 **BESANCON**
28 Rue de la Republique
F-25000 **BESANCON**
Tel: (81) 83-16-22
C,M

Hewlett-Packard France
Bureau de Vente de Lyon
Chemin des Mouilles
Boite Postale No. 162
F-69130 **ECULLY** Cédex
Tel: (78) 33-81-25
Telex: 310617F
A,CM,CP,E,MP

Hewlett-Packard France
Immeuble France Evry
Tour Lorraine
Boulevard de France
F-91035 **EVRY** Cédex
Tel: (60) 77-96-60
Telex: 692315F
CM,E

Hewlett-Packard France
5th Avenue Raymond Chanas
F-38320 **EYBENS**
Tel: (76) 25-81-41
Telex: 980124 HP GRENOB EYBE
CM,CS

Hewlett-Packard France
Bâtiment Ampère
Rue de la Commune de Paris
Boite Postale 300
F-93153 **LE BLANC MESNIL**
Tel: (01) 865-44-52
Telex: 211032F
CM,CP,E,MS

Hewlett-Packard France
Le Montesquieu
Avenue du President JF Kennedy
F-33700 **MERIGNAC**
Tel: (56) 34-00-84
Telex: 550105F
CM,CP,E,MS

Hewlett-Packard France
32 Rue Lothaire
F-57000 **METZ**
Tel: (87) 65-53-50
CM,CS

Hewlett-Packard France
Zone d'activities de Courtaboeuf
Avenue des Tropiques
Boite Postale 6
F-91401 **ORSAY** Cédex
Tel: (1) 907-78-25
Telex: 600048F
A,CM,CP,E,MP,P

Hewlett-Packard France
Paris Porte-Maillot
13, 15 25 Boulevard De L'Amiral
Bruix
F-75782 **PARIS** Cédex 16
Tel: (01) 502-12-20
Telex: 613663F
CM,CP,MS,P

Hewlett-Packard France
2 Allee de la Bourgonette
F-35100 **RENNES**
Tel: (99) 51-42-44
Telex: 740912F
CM,CS,E,MS,P*

Hewlett-Packard France
4 Rue Thomas Mann
Boite Postale 56
F-67200 **STRASBOURG**
Tel: (88) 28-56-46
Telex: 890141F
CM,CS,E,MS,P*

Hewlett-Packard France
20 Chemin de la Cépière
F-31081 **TOULOUSE** Cédex
Tel: (61) 40-11-12
Telex: 531639F
A,CM,CS,E,P*

Hewlett-Packard France
Bureau de Vente de Lille
Immeuble Péricentre
Rue Van Gogh
F-59650 **VILLENEUVE D'ASQ**
Tel: (20) 91-41-25
Telex: 160124F
CM,CS,E,MS,P*

GERMAN FEDERAL REPUBLIC
Hewlett-Packard GmbH
Technisches Büro Berlin
Keithstrasse 2-4
D-1000 **BERLIN** 30
Tel: (030) 24-90-86
Telex: 018 3405 hpbln d
A,CM,CS,E,X,M,P

Hewlett-Packard GmbH
Technisches Büro Böblingen
Herrenberger Strasse 110
D-7030 **BÖBLINGEN**
Tel: (07031) 667-1
Telex: 07265739 bbn or 07265743
A,CM,CP,E,MP,P

Hewlett-Packard GmbH
Technisches Büro Dusseldorf
Emanuel-Leutze-Strasse 1
D-4000 **DUSSELDORF**
Tel: (0211) 5971-1
Telex: 085/86 533 hpdd d
A,CM,CP,E,MS,P

Hewlett-Packard GmbH
Vertriebszentrale Frankfurt
Berner Strasse 117
Postfach 560 140
D-6000 **FRANKFURT** 56
Tel: (0611) 50-04-1
Telex: 04 13249 hpffm d
A,CM,CP,E,MP,P

Hewlett-Packard GmbH
Technisches Büro Hamburg
Kapstadtring 5
D-2000 **HAMBURG** 60
Tel: (040) 63804-1
Telex: 21 63 032 hphh d
A,CM,CP,E,MS,P

Hewlett-Packard GmbH
Technisches Büro Hannover
Am Grossmarkt 6
D-3000 **HANNOVER** 91
Tel: (0511) 46-60-01
Telex: 092 3259
A,CM,CS,E,MS,P

Hewlett-Packard GmbH
Technisches Büro Mannheim
Rosslauer Weg 2-4
D-6800 **MANNHEIM**
Tel: (621) 70050 Telex: 0462105
A,C,E

Hewlett-Packard GmbH
Technisches Büro Neu Ulm
Messerschmittstrasse 7
D-7910 **NEU ULM**
Tel:
Telex:

Hewlett-Packard GmbH
Technisches Büro Nürnberg
Neumeyerstrasse 90
D-8500 **NÜRNBERG**
Tel: (0911) 56-30-83
Telex: 0623 860
CM,CS,E,MS,P

Hewlett-Packard GmbH
Technisches Büro München
Eschenstrasse 5
D-8021 **TAUFKIRCHEN**
Tel: (089) 6117-1
Telex: 0524985
A,CM,CP,E,MS,P

GREAT BRITAIN
Hewlett-Packard Ltd.
Trafalgar House
Navigation Road
ALTRINCHAM
Chesire WA14 1NU
Tel: (061) 928-6422
Telex: 668068
A,C,E,M

Hewlett-Packard Ltd.
Oakfield House, Oakfield Grove
Clifton
BRISTOL BS8 2BN
Tel: 36806
Telex: 444302
P

Hewlett-Packard Ltd.
14 Wesley Street
CASTLEFORD
Yorkshire WF10 1AE
Tel: (0977) 550016
Telex: 5557355
C

Hewlett-Packard Ltd.
Fourier House
257-263 High Street
LONDON COLNEY
Herts., AL2 1HA
Tel: (0727) 24400
Telex: 1-8952716
C,E

Hewlett-Packard Ltd
Tradax House, St. Mary's Walk
MAIDENHEAD
Berkshire, SL6 1ST
Tel: (0628) 39151
E,P

Hewlett-Packard Ltd.
308/314 Kings Road
READING, Berkshire
Tel: 61022
Telex: 84-80-68
CM,P

Hewlett-Packard Ltd.
Quadrangle
106-118 Station Road
REDHILL, Surrey
Tel: (0737) 68655
Telex: 947234
C,E

Hewlett-Packard Ltd.
Westminster House
190 Stratford Road
SHIRLEY, Solihull
West Midlands B90 3BJ
Tel: (021) 7458800
Telex: 339105
C

Hewlett-Packard Ltd.
King Street Lane
WINNERSH, Wokingham
Berkshire RG11 5AR
Tel: (0734) 784774
Telex: 847178
A,C,E,M

GREECE
Kostas Karaynnis
8 Omirou Street
***ATHENS** 133*
Tel: 32-30-303, 32-37-371
Telex: 21 59 62 RKAR GR
E,M,P

"Plaiso"
G. Gerados
24 Stournara Street
ATHENS
Tel: 36-11-160
Telex: 21 9492
P

GUATEMALA
IPESA
Avenida Reforma 3-48
Zona 9
GUATEMALA CITY
Tel: 316627, 314786, 664715
Telex: 4192 Teletro Gu
A,C,E,M,P

HONG KONG
Hewlett-Packard Hong Kong, Ltd.
G.P.O. Box 795
5th Floor, Sun Hung Kai Centre
30 Harbour Road
HONG KONG
Tel: 5-8323211
Telex: 66678 HEWPA HX
Cable: HP ASIA LTD Hong Kong
E,CP,P

Schmidt & Co. (Hong Kong) Ltd.
Wing On Centre, 28th Floor
Connaught Road, C.
HONG KONG
Tel: 5-455644
Telex: 74766 SCHMX HX
A,M

ICELAND
Elding Trading Company Inc.
Hafnarnvoli-Tryggvagotu
P.O. Box 895
IS-REYKJAVIK
Tel: 1-58-20, 1-63-03
M

INDIA
Blue Star Ltd.
Bhavdeep
Stadium Road
***AHMEDABAD** 380 014*
Tel: 42932
Telex: 012-234
Cable: BLUEFROST
E

Blue Star Ltd.
11 Magarath Road
***BANGALORE** 560 025*
Tel: 55668
Telex: 0845-430
Cable: BLUESTAR
A,CM,C,E

Blue Star Ltd.
Band Box House
Prabhadevi
***BOMBAY** 400 025*
Tel: 45-73-01
Telex: 011-3751
Cable: BLUESTAR
A,M

Blue Star Ltd.
Sahas
414/2 Vir Savarkar Marg
Prabhadevi
***BOMBAY** 400 025*
Tel: 46-65-55
Telex: 011-4093
Cable: FROSTBLUE
A,CM,C,E,M

Blue Star Ltd.
7 Hare Street
***CALCUTTA** 700 001*
Tel: 12-01-31
Telex: 021-7655
Cable: BLUESTAR
A,M

Blue Star Ltd.
Meenakshi Mandiram
XXXXV/1379-2 Mahatma Gandhi Road
***COCHIN** 682-016*
Tel: 32069
Telex: 085-514
Cable: BLUESTAR
*A**

Blue Star Ltd.
133 Kodambakkam High Road
***MADRAS** 600 034*
Tel: 82057
Telex: 041-379
Cable: BLUESTAR
A,M

Blue Star Ltd.
Bhandari House, 7th/8th Floors
91 Nehru Place
***NEW DELHI** 110 024*
Tel: 682547
Telex: 031-2463
Cable: BLUESTAR
A,CM,C,E,M

Blue Star Ltd.
1-1-117/1 Sarojini Devi Road
***SECUNDERABAD** 500 033*
Tel: 70126
Telex: 0155-459
Cable: BLUESTAR
A,E

Blue Star Ltd.
T.C. 7/603 Poornima
Maruthankuzhi
***TRIVANDRUM** 695 013*
Tel: 65799
Telex: 0884-259
Cable: BLUESTAR
E

INDONESIA
BERCA Indonesia P.T.
P.O.Box 496/Jkt.
Jln. Abdul Muis 62
JAKARTA
Tel: 373009
Telex: 31146 BERSAL IA
Cable: BERSAL JAKARTA
A,E,M,P

BERCA Indonesia P.T.
P.O. Box 174/Sby.
J.L. Kutei No. 11
SUBAEE-SURABAYA
Tel: 68172
Telex: 31146 BERSAL SD
Cable: BERSAL-SURABAYA
A,E,M,P*

IRAQ
Hewlett-Packard Trading S.A.
Mansoor City 9B/3/7
BAGHDAD
Tel: 551-49-73
Telex: 2455 HEPAIRAQ IK
CP

IRELAND
Hewlett-Packard Ireland Ltd.
Kestrel House
Clanwilliam Court
Lower Mount Street
DUBLIN 2, Eire
Tel: 680424, 680426
Telex: 30439
A,C,CM,E,M,P

Cardiac Services Ltd.
Kilmore Road
Artane
***DUBLIN** 5, Eire*
Tel: (01) 351820
Telex: 30439
M

ISRAEL
Electronics Engineering Division
Motorola Israel Ltd.
16 Kremenetski Street
P.O. Box 25016
TEL-AVIV 67899
Tel: 38973
Telex: 33569 Motil IL
Cable: BASTEL Tel-Aviv
A,CM,C,E,M,P

ITALY
Hewlett-Packard Italiana S.p.A.
Traversa 99C
Giulio Petrone, 19
I-70124 **BARI**
Tel: (080) 41-07-44
M

Hewlett-Packard Italiana S.p.A.
Via Martin Luther King, 38/111
I-40132 **BOLOGNA**
Tel: (051) 402394
Telex: 511630
CM,CS,E,MS

Hewlett-Packard Italiana S.p.A.
Via Principe Nicola 43G/C
I-95126 **CATANIA**
Tel: (095) 37-10-87 Telex: 970291
C,P

Hewlett-Packard Italiana S.p.A.
Via G. Di Vittorio 9
I-20063 **CERNUSCO SUL NAVLIGLIO**
Tel: (2) 903691
Telex: 334632
A,CM,CP,E,MP,P

Hewlett-Packard Italiana S.p.A.
Via Nuova san Rocco A
Capodimonte, 62/A
I-80131 **NAPOLI**
Tel: (081) 7413544
A,CM,CS,E

Hewlett-Packard Italiana S.p.A.
Viale G. Modugno 33
I-16156 **GENOVA PEGLI**
Tel: (010) 68-37-07
E,C

Hewlett-Packard Italiana S.p.A.
Via Turazza 14
I-35100 **PADOVA**
Tel: (49) 664888
Telex: 430315
A,CM,CS,E,MS

Hewlett-Packard Italiana S.p.A.
Viale C. Pavese 340
I-00144 **ROMA**
Tel: (06) 54831
Telex: 610514
A,CM,CS,E,MS,P*

Hewlett-Packard Italiana S.p.A.
Corso Giovanni Lanza 94
I-10133 **TORINO**
Tel: (011) 682245, 659308
Telex: 221079
CM,CS,E

JAPAN
Yokogawa-Hewlett-Packard Ltd.
Inoue Building
1348-3, Asahi-cho
ATSUGI, Kanagawa 243
Tel: (0462) 24-0451
CM,C*,E

Yokogawa-Hewlett-Packard Ltd.
3-30-18 Tsuruya-cho
Kanagawa-ku, Yokohama-Shi
KANAGAWA, 221
Tel: (045) 312-1252
Telex: 382-3204 YHP YOK
CM,CS,E

Yokogawa-Hewlett-Packard Ltd.
Sannomiya-Daiichi Seimei-Bldg. 5F
69 Kyo-Machi Ikuta-Ku
KOBE CITY 650 Japan
Tel: (078) 392-4791
C,E

Yokogawa-Hewlett-Packard Ltd.
Kumagaya Asahi Yasoji Bldg 4F
4-3 Chome Tsukuba
KUMAGAYA, Saitama 360
Tel: (0485) 24-6563
CM,CS,E

Yokogawa-Hewlett-Packard Ltd.
Mito Mitsui Building
4-73, San-no-maru, 1-chome
MITO, Ibaragi 310
Tel: (0292) 25-7470
CM,CS,E

Yokogawa-Hewlett-Packard Ltd.
Sumitomo Seimei Bldg.
11-2 Shimo-sasajima-cho
Nakamura-ku
NAGOYA, Aichi 450
Tel: (052) 581-1850
CM,CS,E,MS

Yokogawa-Hewlett-Packard Ltd.
Chuo Bldg., 4th Floor
5-4-20 Nishinakajima, 5-chome
Yodogawa-ku, Osaka-shi
OSAKA, 532
Tel: (06) 304-6021
Telex: YHPOSA 523-3624
A,CM,CP,E,MP,P*

Yokogawa-Hewlett-Packard Ltd.
29-21 Takaido-Higashi 3-chome
Suginami-ku **TOKYO** 168
Tel: (03) 331-6111
Telex: 232-2024 YHPTOK
A,CM,CP,E,MP,P*

JORDAN
Mouasher Cousins Company
P.O. Box 1387
AMMAN
Tel: 24907, 39907
Telex: 21456 SABCO JO
E,M,P

KOREA
Samsung Electronics
4759 Singil, 6 Dong
Young Deung Po Ku,
SEOUL
Tel: 8334311, 8330002
Telex: SAMSAN 27364
A,C,E,M,P

KUWAIT
Al-Khalidya Trading & Contracting
P.O. Box 830 Safat
KUWAIT
Tel: 42-4910, 41-1726
Telex: 2481 Areeg kt
A,E,M

Photo & Cine Equipment
P.O. Box 270 Safat
KUWAIT
Tel: 42-2846, 42-3801
Telex: 2247 Matin
P

LUXEMBOURG
Hewlett-Packard Belgium S.A./N.V.
Blvd de la Woluwe, 100
Woluwedal
*B-1200 **BRUSSELS***
Tel: (02) 762-32-00
Telex: 23-494 paloben bru
A,CM,CP,E,MP,P

MALAYSIA
Hewlett-Packard Sales (Malaysia)
Sdn. Bhd.
Suite 2.21/2.22
Bangunan Angkasa Raya
Jalan Ampang
KUALA LUMPUR
Tel: 483544
Telex: MA31011
A,CP,E,M,P*

Protel Engineering
P.O. Box 1917
Lot 319, Satok Road
*Kuching, **SARAWAK***
Tel: 53544
Telex: MA 70904 PROMAL
Cable: PROTELENG
A,E,M

MEXICO
Hewlett-Packard Mexicana, S.A. de
C.V.
Avenida Periferico Sur No. 6501
Tepepan, Xochimilco
MEXICO CITY 23, D.F.
Tel: (905) 676-4600
Telex: 017-74-507
A,CP,E,MS,P

Hewlett-Packard Mexicana, S.A. de
C.V.
Rio Volga 600
Colonia del Valle
MONTERREY, N.L.
Tel: 78-42-93, 78-42-40, 78-42-41
Telex: 038-410
CS

MOROCCO
Dolbeau
81 rue Karatchi
CASABLANCA
Tel: 3041-82, 3068-38
Telex: 23051, 22822
E

Gerep
2 rue d'Agadir
Boite Postale 156
CASABLANCA
Tel: 272093, 272095
Telex: 23 739
P

MOZAMBIQUE
A.N. Goncalves Ltd.
162, 1 Apt. 14 Av. D. Luis
Caixa Postal 107
MAPUTO
Tel: 27091, 27114
Telex: 6-203 NEGON Mo
Cable: NEGON
A,E,M,P

NETHERLANDS
Hewlett-Packard Nederland B.V.
Van Heuven Goedhartlaan 121
NL 1181KK **AMSTELVEEN**
P.O. Box 667
NL1080 AR **AMSTELVEEN**
Tel: (20) 47-20-21
Telex: 13 216
A,CM,CP,E,MP,P

Hewlett-Packard Nederland B.V.
Bongerd 2
NL 2906VK **CAPPELLE**, A/D Ijessel
P.O. Box 41
NL2900 AA **CAPELLE**, Ijssel
Tel: (10) 51-64-44
Telex: 21261 HEPAC NL
A,CM,CP

NEW ZEALAND
Hewlett-Packard (N.Z.) Ltd.
169 Manukau Road
P.O. Box 26-189
Epsom, **AUCKLAND**
Tel: 68-7159
Cable: HEWPACK Auckland
CM,CS,E,P*

Hewlett-Packard (N.Z.) Ltd.
4-12 Cruickshank Street
P.O. Box 9443
Kilbirnie, **WELLINGTON** 3
Tel: 877-199
Cable: HEWPACK Wellington
CM,CP,E,P

Northrop Instruments & Systems
Ltd.
Eden House, 44 Khyber Pass Road
P.O. Box 9682
*Newmarket, **AUCKLAND***
Tel: 794-091
A,M

Northrop Instruments & Systems
Ltd.
Terrace House, 4 Oxford Terrace
P.O. Box 8388
CHRISTCHURCH
Tel: 64-165
A,M

Northrop Instruments & Systems
Ltd.
Sturdee House
85-87 Ghuznee Street
P.O. Box 2406
WELLINGTON
Tel: 850-091
Telex: NZ 3380
A,M

NIGERIA
The Electronics Instrumentations
Ltd.
N6B/S70 Oyo Road
Oluseun House
P.M.B. 5402
IBADAN
Tel: 461577
Telex: 31231 TEIL NG
A,E,M,P

The Electronics Instrumentations
Ltd.
144 Agege Motor Road, Mushin
P.O. Box 6645
*Mushin, **LAGOS***
A,E,M,P

NORTHERN IRELAND
Cardiac Services Company
95A Finaghy Road South
***BELFAST** BT 10 OBY*
Tel: (0232) 625-566
Telex: 747626
M

NORWAY
Hewlett-Packard Norge A/S
Folke Bernadottesvei 50
P.O. Box 3558
N-5033 **FYLLINGSDALEN** (BERGEN)
Tel: (05) 16-55-40
Telex: 16621 hpnas n
CM,CS,E

Hewlett-Packard Norge A/S
Oestendalen 18
P.O. Box 34
N-1345 **OESTERAAS**
Tel: (02) 17-11-80
Telex: 16621 hpnas n
A*,CM,CP,E,MS,P

OMAN
Khimjil Ramdas
P.O. Box 19
MUSCAT
Tel: 72-22-17, 72-22-25
Telex: 3289 BROKER MB MUSCAT
P

PAKISTAN
Mushko & Company Ltd.
10, Bazar Road
Sector G-6/4
ISLAMABAD
Tel: 28624
Cable: FEMUS Rawalpindi
A,E,M

Mushko & Company Ltd.
Oosman Chambers
Abdullah Haroon Road
***KARACHI** 0302*
Tel: 511027, 512927
Telex: 2894 MUSHKO PW
Cable: COOPERATOR Karachi
*A,E,M,P**

PANAMA
Electrónico Balboa, S.A.
Apartado 4929
Panama 5
Calle Samuel Lewis
Edificio "Alfa" No. 2
CIUDAD DE PANAMA
Canal Zone
Tel: 64-2700
Telex: 3480380
Cable: ELECTRON Panama
A,E,M

Foto Internacional, S.A.
P.O. Box 2068
Free Zone of Colon
COLON 3
Tel: 45-2333
Telex: 3485126
Cable: IMPORT COLON/Panama
P

PERU
Cómpania Electro Médica
S.A.
Los Flamencos 145, San Isidro
Casilla 1030
***LIMA** 1*
Tel: 41-4325
Telex: Pub. Booth 25424 SISIDRO
Cable: ELMED Lima
A,E,M

PHILIPPINES
The Online Advanced Systems
Corporation
Rico House, Amorsolo Cor. Herrera
Street
Legaspi Village, Makati
P.O. Box 1510
*Metro **MANILA***
Tel: 85-35-81, 85-34-91, 85-32-21
Telex: 3274 ONLINE
A,C,E,M

SALES & SUPPORT OFFICES
Arranged alphabetically by country

PHILIPPINES (Cont'd)
Electronic Specialists and Proponents Inc.
690-B Epifanio de los Santos Avenue
Cubao, **QUEZON CITY**
P.O. Box 2649 **Manila**
Tel: 98-96-81, 98-96-82, 98-96-83
Telex: 742-40287
P

POLAND
Buro Informasji Technicznej
Hewlett-Packard
Ul Stawki 2, 6P
PL00-950 **WARSZAWA**
Tel: 39-59-62, 39-67-43
Telex: 812453 hepa pl

PORTUGAL
Telectra-Empresa Técnica de Equipmentos Eléctricos S.a.r.l.
Rua Rodrigo da Fonseca 103
P.O. Box 2531
P-LISBON 1
Tel: (19) 68-60-72
Telex: 12598
A,C,E,P

Mundinter
Intercambio Mundial de Comércio S.a.r.l
P.O. Box 2761
Avenida Antonio Augusto de Aguiar 138
P-LISBON
Tel: (19) 53-21-31, 53-21-37
Telex: 16691 munter p
M

PUERTO RICO
Hewlett-Packard Puerto Rico
P.O. Box 4407
CAROLINA, Puerto Rico 00630
Calle 272 Edificio 203
Urb. Country Club
RIO PIEDRAS, Puerto Rico 00924
Tel: (809) 762-7255
Telex: 345 0514
A,CP

QATAR
Nasser Trading & Contracting
P.O. Box 1563
DOHA
Tel: 22170
Telex: 4439 NASSER
M

Scitecharabia
P.O. Box 2750
!'DOHA
Tel: 329515
Telex: 4806 CMPARB
P

ROMANIA
Hewlett-Packard Reprezentanta
Boulevard Nicolae Balcescu 16
BUCURESTI
Tel: 130725
Telex: 10440

SAUDI ARABIA
Modern Electronic Establishment
P.O. Box 193
AL-KHOBAR
Tel: 44-678, 44-813
Telex: 670136
Cable: ELECTA AL-KHOBAR
C,E,M,P

Modern Electronic Establishment
P.O. Box 1228, Baghdadiah Street
JEDDAH
Tel: 27-798
Telex: 401035
Cable: ELECTA JEDDAH
C,E,M,P

Modern Electronic Establishment
P.O. Box 2728
RIYADH
Tel: 62-596, 66-232
Telex: 202049
C,E,M,P

SCOTLAND
Hewlett-Packard Ltd.
Royal Bank Buildings
Swan Street
BRECHIN, Angus, Scotland
Tel: 3101, 3102
CM,CS

Hewlett-Packard Ltd.
SOUTH QUEENSFERRY
West Lothian, EH30 9TG
GB-Scotland
Tel: (031) 3311000
Telex: 72682
A,CM,E,M

SINGAPORE
Hewlett-Packard Singapore (Pty.) Ltd.
P.O. Box 58 Alexandra Post Office
SINGAPORE, 9115
6th Floor, Inchcape House
450-452 Alexandra Road
SINGAPORE 0511
Tel: 631788
Telex: HPSGSO RS 34209
Cable: HEWPACK, Singapore
A,CP,E,MS,P

SOUTH AFRICA
Hewlett-Packard South Africa (Pty.) Ltd.
P.O. Box 120
Howard Place
Pine Park Center, Forest Drive, Pinelands
CAPE PROVINCE 7450
Tel: 53-7955, 53-7956, 53-7957
Telex: 57-0006
A,CM,CS,E,MS,P

Hewlett-Packard South Africa (Pty.) Ltd.
P.O. Box 37066
Overport
DURBAN 4067
Tel: 28-4178, 28-4179, 28-4110
CM,CS

Hewlett-Packard South Africa (Pty.) Ltd.
Daphny Street
Private Bag Wendywood
SANDTON 2144
Tel: 802-5111, 802-5125
Telex: 89-84782
Cable: HEWPACK Johannesburg
A,CM,CP,E,MS,P

SPAIN
Hewlett-Packard Española S.A.
c/Entenza, 321
E-BARCELONA 29
Tel: (3) 322-24-51, 321-73-54
Telex: 52603 hpbee
A,CM,CP,E,MS,P

Hewlett-Packard Española S.A.
c/San Vicente S/N
Edificio Albia II,7 B
E-BILBAO 1
Tel: (944) 423-8306, 423-8206
A,CM,E,MS

Hewlett-Packard Española S.A.
Calle Jerez 3
E-MADRID 16
Tel: 458-2600
Telex: 23515 hpe
A,CM,E,MP,P

Hewlett-Packard Española S.A.
Colonia Mirasierra
Edificio Juban
c/o Costa Brava 13, 2.
E-MADRID 34
Tel: 734-8061, 734-1162
CM,CP

Hewlett-Packard Española S.A.
Av Ramón y Cajal 1-9
Edificio Sevilla 1,
E-SEVILLA 5
Tel: 64-44-54, 64-44-58
Telex: 72933
A,CM,CS,MS,P

Hewlett-Packard Española S.A.
C/Ramon Gordillo, 1 (Entlo.3)
E-VALENCIA 10
Tel: 361-1354, 361-1358
CM,CS,P

SWEDEN
Hewlett-Packard Sverige AB
Enighetsvägen 3, Fack
P.O. Box 20502
S-16120 **BROMMA**
Tel: (08) 730-0550
Telex: (854) 10721 MESSAGES
Cable: MEASUREMENTS STOCKHOLM
A,CM,CP,E,MS,P

Hewlett-Packard Sverige AB
Sunnanvagen 14K
S-22226 **LUND**
Tel: (46) 13-69-79
Telex: (854) 10721 (via BROMMA office)
CM,CS

Hewlett-Packard Sverige AB
Vastra Vintergatan 9
S-70344 **OREBRO**
Tel: (19) 10-48-80
Telex: (854) 10721 (via BROMMA office)
CM,CS

Hewlett-Packard Sverige AB
Frötallisgatan 30
S-42132 **VÄSTRA-FRÖLUNDA**
Tel: (031) 49-09-50
Telex: (854) 10721 (via BROMMA office)
CM,CS,E,P

SWITZERLAND
Hewlett-Packard (Schweiz) AG
Clarastrasse 12
CH-4058 **BASLE**
Tel: (61) 33-59-20
A,CM

Hewlett-Packard (Schweiz) AG
47 Avenue Blanc
CH-1202 **GENEVA**
Tel: (022) 32-30-05, 32-48-00
CM,CP

Hewlett-Packard (Schweiz) AG
29 Chemin Château Bloc
CH-1219 **LE LIGNON**-Geneva
Tel: (022) 96-03-22
Telex: 27333 hpag ch
Cable: HEWPACKAG Geneva
A,CM,E,MS,P

Hewlett-Packard (Schweiz) AG
Zürcherstrasse 20
Allmend 2
CH-8967 **WIDEN**
Tel: (57) 50-111
Telex: 59933 hpag ch
Cable: HPAG CH
A,CM,CP,E,MS,P

SYRIA
General Electronic Inc.
Nuri Basha-Ahnat Ebn Kays Street
P.O. Box 5781
DAMASCUS
Tel: 33-24-87
Telex: 11215 ITIKAL
Cable: ELECTROBOR DAMASCUS
E

Sawah & Co.
Place Azmé
Boite Postale 2308
DAMASCUS
Tel: 16-367, 19-697, 14-268
Telex: 11304 SATACO SY
Cable: SAWAH, DAMASCUS
M

TAIWAN
Hewlett-Packard Far East Ltd.
Kaohsiung Branch
68-2, Chung Cheng 3rd Road
Shin Shin, Chu
KAOHSIUNG
Tel: 24-2318, 26-3253
CS,E,MS,P

Hewlett-Packard Far East Ltd.
Taiwan Branch
5th Floor
205 Tun Hwa North Road
TAIPEI
Tel:(02) 751-0404
Cable:HEWPACK Taipei
A*,CP,E,MS,P

Hewlett-Packard Far East Ltd.
Taichung Branch
#33, Cheng Yih Street
10th Floor, Room 5
TAICHUNG
Tel: 289274

San Kwang Instruments Co., Ltd.
20 Yung Sui Road
TAIPEI
Tel: 361-5446, 361-5447, 361-5448
Telex: 22894 SANKWANG
Cable: SANKWANG Taipei
A

THAILAND
UNIMESA Co. Ltd.
Elcom Research Building
2538 Sukhumvit Ave.
Bangchak, **BANGKOK**
Tel: 393-2387, 393-0338
Telex: TH81160, 82938, 81038
Cable: UNIMESA Bangkok
A,E,M

Bangkok Business Equipment Ltd.
5/5-6 Dejo Road
BANGKOK
Tel: 234-8670, 234-8671, 234-8672
Cable: BUSIQUIPT Bangkok
P

TRINIDAD & TOBAGO
CARTEL
Caribbean Telecoms Ltd.
P.O. Box 732
50/A Jerningham Avenue
PORT-OF-SPAIN
Tel: 624-4213, 624-4214
A,E,M,P

TUNISIA
Tunisie Electronique
31 Avenue de la Liberte
TUNIS
Tel: 280-144
E,P

Corema
1 ter. Av. de Carthage
TUNIS
Tel: 253-821
Telex: 12319 CABAM TN
M

TURKEY
Teknim Company Ltd.
Riza Sah Pehlevi
Caddesi No. 7
Kavaklidere, **ANKARA**
Tel: 275800
Telex: 42155
E

EMA, Muhendislik Kollektif Sirketi
Mediha Eldem
Sokak 41/6
Yüksel Caddesi, **ANKARA**
Tel: 17-56-22
Cable: Ematrade
M

UNITED ARAB EMIRATES
Emitac Ltd.
P.O. Box 1641
SHARJAH
Tel: 354121, 354123
Telex: 68136
E,M,P,C

UNITED KINGDOM
see: GREAT BRITAIN
NORTHERN IRELAND
SCOTLAND

UNITED STATES

Alabama
Hewlett-Packard Co.
700 Century Park South
Suite 128
BIRMINGHAM, AL 35226
Tel: (205) 822-6802
CM,CS,MP

Hewlett-Packard Co.
P.O. Box 4207
8290 Whitesburg Drive, S.E.
HUNTSVILLE, AL 35802
Tel: (205) 881-4591
CM,CP,E,M*

Alaska
Hewlett-Packard Co.
1577 "C" Street, Suite 252
ANCHORAGE, AK 99510
Tel: (206) 454-3971
CM,CS**

Arizona
Hewlett-Packard Co.
2336 East Magnolia Street
PHOENIX, AZ 85034
Tel: (602) 273-8000
A,CM,CP,E,MS

Hewlett-Packard Co.
2424 East Aragon Road
TUCSON, AZ 85702
Tel: (602) 889-4631
CM,CS,E,MS**

Arkansas
Hewlett-Packard Co.
P.O. Box 5646
Brady Station
LITTLE ROCK, AR 72215
Tel: (501) 376-1844, (501) 664-8773
CM,MS

UNITED STATES (Cont'd)

California
Hewlett-Packard Co.
7621 Canoga Avenue
CANOGA PARK, CA 91304
Tel: (213) 702-8300
A,CM,CP,E,P

Hewlett-Packard Co.
1579 W. Shaw Avenue
FRESNO, CA 93771
Tel: (209) 224-0582
CM,MS

Hewlett-Packard Co.
1430 East Orangethorpe
FULLERTON, CA 92631
Tel: (714) 870-1000
CM,CP,E,MP

Hewlett-Packard Co.
5400 W. Rosecrans Boulevard
LOS ANGELES, CA 90260
Tel: (213) 970-7500
CM,CP,MP

Hewlett-Packard Co.
3939 Lankershim Blvd.
NORTH HOLLYWOOD, CA 91604
Tel: (213) 877-1282
regional headquarters

Hewlett-Packard Co.
3200 Hillview Avenue
PALO ALTO, CA 94304
Tel: (415) 857-8000
CM,CP,E

Hewlett-Packard Co.
646 W. North Market Boulevard
SACRAMENTO, CA 95834
Tel: (916) 929-7222
A*,CM,CP,E,MS

Hewlett-Packard Co.
9606 Aero Drive
P.O. Box 23333
SAN DIEGO, CA 92123
Tel: (714) 279-3200
CM,CP,E,MP

Hewlett-Packard Co.
3003 Scott Boulevard
SANTA CLARA, CA 95050
Tel: (408) 988-7000
A,CM,CP,E,MP

Hewlett-Packard Co.
454 Carlton Court
SO. SAN FRANCISCO, CA 94080
Tel: (415) 877-0772
CM,CP

Colorado
Hewlett-Packard Co.
24 Inverness Place, East
ENGLEWOOD, CO 80112
Tel: (303) 771-3455
A,CM,CP,E,MS

Connecticut
Hewlett-Packard Co.
47 Barnes Industrial Road South
P.O. Box 5007
WALLINGFORD, CT 06492
Tel: (203) 265-7801
A,CM,CP,E,MS

Florida
Hewlett-Packard Co.
P.O. Box 24210
2727 N.W. 62nd Street
FORT LAUDERDALE, FL 33309
Tel: (305) 973-2600
CM,CP,E,MP

Hewlett-Packard Co.
4080 Woodcock Drive, #132
Brownett Building
JACKSONVILLE, FL 32207
Tel: (904) 398-0663
CM,C*,E*,MS**

Hewlett-Packard Co.
P.O. Box 13910
6177 Lake Ellenor Drive
ORLANDO, FL 32809
Tel: (305) 859-2900
A,CM,CP,E,MS

Hewlett-Packard Co.
6425 N. Pensacola Blvd.
Suite 4, Building 1
PENSACOLA, FL 32575
Tel: (904) 476-8422
A,CM,MS

Hewlett-Packard Co.
110 South Hoover, Suite 120
Vanguard Bldg.
TAMPA, FL 33609
Tel: (813) 872-0900
A*,CM,CS,E*,M*

Georgia
Hewlett-Packard Co.
P.O. Box 105005
2000 South Park Place
ATLANTA, GA 30339
Tel: (404) 955-1500
Telex: 810-766-4890
A,CM,CP,E,MP

Hewlett-Packard Co.
Executive Park Suite 306
P.O. Box 816
AUGUSTA, GA 30907
Tel: (404) 736-0592
CM,MS

Hewlett-Packard Co.
P.O. Box 2103
1172 N. Davis Drive
WARNER ROBINS, GA 31098
Tel: (912) 922-0449
CM,E

Hawaii
Hewlett-Packard Co.
Kawaiahao Plaza, Suite 190
567 South King Street
HONOLULU, HI 96813
Tel: (808) 526-1555
A,CM,CS,E,MS

Idaho
Hewlett-Packard Co.
11311 Chinden Boulevard
BOISE, ID 83707
Tel: (208) 376-6000
CM,CS,M*

Illinois
Hewlett-Packard Co.
211 Prospect Road
BLOOMINGTON, IL 61701
Tel: (309) 663-0383
CM,CS,MS**

Hewlett-Packard Co.
1100 31st Street
DOWNERS GROVE, IL 60515
Tel: (312) 960-5760
CM,CP

Hewlett-Packard Co.
5201 Tollview Drive
ROLLING MEADOWS, IL 60008
Tel: (312) 255-9800
A,CM,CP,E,MP

Indiana
Hewlett-Packard Co.
P.O. Box 50807
7301 No. Shadeland Avenue
INDIANAPOLIS, IN 46250
Tel: (317) 842-1000
A,CM,CS,E,MS

Iowa
Hewlett-Packard Co.
2415 Heinz Road
IOWA CITY, IA 52240
Tel: (319) 351-1020
CM,CS,E*,MS

Kansas
Hewlett-Packard Co.
514 South Westview
P.O. Box 159
DERBY, KA 67037
Tel: (316) 265-5200
CM,CS

Kentucky
Hewlett-Packard Co.
10170 Linn Station Road
Suite 525
LOUISVILLE, KY 40223
Tel: (502) 426-0100
A,CM,CS,MS

Louisiana
Hewlett-Packard Co.
P.O. Box 1449
3229 Williams Boulevard
KENNER, LA 70062
Tel: (504) 443-6201
A,CM,CS,E,MS

Maryland
Hewlett-Packard Co.
7121 Standard Drive
HANOVER, MD 21076
Tel: (301) 796-7700
A,CM,CP,E,MS

Hewlett-Packard Co.
2 Choke Cherry Road
ROCKVILLE, MD 20850
Tel: (301) 948-6370
Telex: 710-828-9685
A,CM,CP,E,MP

Massachusetts
Hewlett-Packard Co.
32 Hartwell Avenue
LEXINGTON, MA 02173
Tel: (617) 861-8960
A,CM,CP,E,MP

Michigan
Hewlett-Packard Co.
23855 Research Drive
FARMINGTON HILLS, MI 48024
Tel: (313) 476-6400
A,CM,CP,E,MP

Hewlett-Packard Co.
4326 Cascade Road S.E.
GRAND RAPIDS, MI 49506
Tel: (616) 957-1970
CM,CS,MS

Minnesota
Hewlett-Packard Co.
2025 W. Larpenteur Ave.
ST. PAUL, MN 55113
Tel: (612) 644-1100
A,CM,CP,E,MP

Mississippi
Hewlett-Packard Co.
P.O. Box 5028
322 N. Mart Plaza
JACKSON, MS 39216
Tel: (601) 982-9363
CM,MS

Missouri
Hewlett-Packard Co.
11131 Colorado Avenue
KANSAS CITY, MO 64137
Tel: (816) 763-8000
Telex: 910-771-2087
A,CM,CS,E,MS

Hewlett-Packard Co.
1024 Executive Parkway
ST. LOUIS, MO 63141
Tel: (314) 878-0200
A,CM,CP,E,MP

Nebraska
Hewlett-Packard
7101 Mercy Road
Suite 101, IBX Building
OMAHA, NE 68106
Tel: (402) 392-0948
CM,MS

Nevada
Hewlett-Packard Co.
Suite D-130
5030 Paradise Blvd.
LAS VEGAS, NV 89119
Tel: (702) 736-6610
CM,MS**

New Jersey
Hewlett-Packard Co.
Crystal Brook Professional Building
Route 35
EATONTOWN, NJ 07724
Tel: (201) 542-1384
A*,CM,C*,E*,P*

Hewlett-Packard Co.
W120 Century Road
PARAMUS, NJ 07652
Tel: (201) 265-5000
A,CM,CP,E,MP

Hewlett-Packard Co.
60 New England Avenue West
PISCATAWAY, NJ 08854
Tel: (201) 981-1199
A,CM,CP,E

New Mexico
Hewlett-Packard Co.
P.O. Box 11634
11300 Lomas Blvd.,N.E.
ALBUQUERQUE, NM 87123
Tel: (505) 292-1330
Telex: 910-989-1185
CM,CP,E,MS

New York
Hewlett-Packard Co.
5 Computer Drive South
ALBANY, NY 12205
Tel: (518) 458-1550
Telex: 710-444-4691
A,CM,CS,E,MS

Hewlett-Packard Co.
9600 Main Street
CLARENCE, NY 14031
Tel: (716) 759-8621
Telex: 710-523-1893

Hewlett-Packard Co.
200 Cross Keys Office
FAIRPORT, NY 14450
Tel: (716) 223-9950
Telex: 510-253-0092
CM,CP,E,MS

Hewlett-Packard Co.
No. 1 Pennsylvania Plaza
55th Floor
34th Street & 8th Avenue
NEW YORK, NY 10119
Tel: (212) 971-0800
CM,CP,E*,M*

Hewlett-Packard Co.
5858 East Molloy Road
SYRACUSE NY 13211
Tel: (315) 455-2486
A,CM,CS,E,MS

Hewlett-Packard Co.
3 Crossways Park West
WOODBURY, NY 11797
Tel: (516) 921-0300
Telex: 510-221-2183
A,CM,CP,E,MS

North Carolina
Hewlett-Packard Co.
P.O. Box 15579
2905 Guess Road (27705)
DURHAM, NC 27704
Tel: (919) 471-8466
C,M

Hewlett-Packard Co.
5605 Roanne Way
GREENSBORO, NC 27409
Tel: (919) 852-1800
A,CM,CP,E,MS

Ohio
Hewlett-Packard Co.
9920 Carver Road
CINCINNATI, OH 45242
Tel: (513) 891-9870
CM,CP,MS

Hewlett-Packard Co.
16500 Sprague Road
CLEVELAND, OH 44130
Tel: (216) 243-7300
Telex: 810-423-9430
A,CM,CP,E,MS

Hewlett-Packard Co.
962 Crupper Ave.
COLUMBUS, OH 43229
Tel: (614) 436-1041
CM,CP,E*

Hewlett-Packard Co.
330 Progress Rd.
DAYTON, OH 45449
Tel: (513) 859-8202
A,CM,CP,E*,MS

Oklahoma
Hewlett-Packard Co.
P.O. Box 366
1503 W. Gore Blvd., Suite #2
LAWTON, OK 73502
Tel: (405) 248-4248
C

Hewlett-Packard Co.
P.O. Box 32008
304 N. Meridan Avenue, Suite A
OKLAHOMA CITY, OK 73107
Tel: (405) 946-9499
A*,CM,CP,E*,MS

Hewlett-Packard Co.
Suite 121
9920 E. 42nd Street
TULSA, OK 74145
Tel: (918) 665-3300
A**,CM,CS,M*

Oregon
Hewlett-Packard Co.
1500 Valley River Drive, Suite 330
EUGENE, OR 97401
Tel: (503) 683-8075
C

Hewlett-Packard Co.
9255 S. W. Pioneer Court
WILSONVILLE, OR 97070
Tel: (503) 682-8000
A,CM,CP,E*,MS

Pennsylvania
Hewlett-Packard Co.
1021 8th Avenue
King of Prussia Industrial Park
KING OF PRUSSIA, PA 19406
Tel: (215) 265-7000
Telex: 510-660-2670
A,CM,CP,E,MP

Hewlett-Packard Co.
111 Zeta Drive
PITTSBURGH, PA 15238
Tel: (412) 782-0400
A,CM,CP,E,MP

SALES & SUPPORT OFFICES
Arranged alphabetically by country

UNITED STATES (Cont'd)

South Carolina
Hewlett-Packard Co.
P.O. Box 6442
6941-0 N. Trenholm Road
COLUMBIA, SC 29260
Tel: (803) 782-6493
CM,CS,E,MS

Hewlett-Packard Co.
814 Wade Hampton Blvd.
Suite 10
GREENVILLE, SC 29609
Tel: (803) 232-0917
C

Tennessee
Hewlett-Packard Co.
P.O. Box 32490
224 Peters Road
Suite 102
KNOXVILLE, TN 37922
Tel: (615) 691-2371
A*,CM,MS

Hewlett-Packard Co.
3070 Directors Row
MEMPHIS, TN 38131
Tel: (901) 346-8370
A,CM,CS,MS

Hewlett-Packard Co.
Suite 103
478 Craighead Street
NASHVILLE, TN 37204
Tel: (615) 383-9136
CM,MS**

Texas
Hewlett-Packard Co.
Suite 310W
7800 Shoalcreek Blvd.
AUSTIN, TX 78757
Tel: (512) 459-3143
CM,E

Hewlett-Packard Co.
Suite C-110
4171 North Mesa
EL PASO, TX 79902
Tel: (915) 533-3555
CM,CS,E*,MS**

Hewlett-Packard Co.
5020 Mark IV Parkway
FORT WORTH, TX 76106
Tel: (817) 625-6361
CM,C*

Hewlett-Packard Co.
P.O. Box 42816
10535 Harwin Street
HOUSTON, TX 77036
Tel: (713) 776-6400
A,CM,CP,E,MP

Hewlett-Packard Co.
3309 67th Street
Suite 24
LUBBOCK, TX 79413
Tel: (806) 799-4472
M

Hewlett-Packard Co.
P.O. Box 1270
930 E. Campbell Rd.
RICHARDSON, TX 75081
Tel: (214) 231-6101
A,CM,CP,E,MP

Hewlett-Packard Co.
205 Billy Mitchell Road
SAN ANTONIO, TX 78226
Tel: (512) 434-8241
CM,CS,E,MS

Utah
Hewlett-Packard Co.
3530 W. 2100 South Street
SALT LAKE CITY, UT 84119
Tel: (801) 974-1700
A,CM,CP,E,MS

Virginia
Hewlett-Packard Co.
P.O. Box 9669
2914 Hungary Spring Road
RICHMOND, VA 23228
Tel: (804) 285-3431
A,CM,CP,E,MS

Hewlett-Packard Co.
P.O. Box 4786
3110 Peters Creek Road, N.W.
ROANOKE, VA 24015
Tel: (703) 922-7000
CM,CS,E**

Hewlett-Packard Co.
P.O. Box 12778
5700 Thurston Avenue
Suite 111
VIRGINIA BEACH, VA 23455
Tel: (804) 460-2471
CM,CS,MS

Washington
Hewlett-Packard Co.
15815 S.E. 37th Street
BELLEVUE, WA 98006
Tel: (206) 643-4000
A,CM,CP,E,MP

Hewlett-Packard Co.
Suite A
708 North Argonne Road
SPOKANE, WA 99206
Tel: (509) 922-7000
CM,CS

West Virginia
Hewlett-Packard Co.
4604 MacCorkle Ave., S.E.
CHARLESTON, WV 25304
Tel: (304) 925-0492
A,CM,MS

Wisconsin
Hewlett-Packard Co.
150 S. Sunny Slope Road
BROOKFIELD, WI 53005
Tel: (414) 784-8800
A,CM,CS,E*,MP

URUGUAY
Pablo Ferrando S.A.C. e.l.
Avenida Italia 2877
Casilla de Correo 370
MONTEVIDEO
Tel: 403102
Telex: 901 Public Booth Para Pablo Ferrando 919520
Cable: RADIUM Montevideo
A,E,M

Guillermo Kraft del Uruguay S.A.
Avda. Libertador Brig. Gral.
Lavalleja 2083
MONTEVIDEO
Tel: 234588, 234808, 208830
Telex: 6245 ACTOUR UY
P

U.S.S.R.
Hewlett-Packard Co.
Representative Office
Pokrovsky Blvd. 4/17 KV12
MOSCOW 101000
Tel: 294-2024
Telex: 7825 HEWPACK SU

VENEZUELA
Hewlett-Packard de Venezuela C.A.
Apartado 50933
3A Transversal Los Ruices Norte
Edificio Segre 2Y3
CARACAS 1071
Tel: 239-4133, 239-4777, 239-4244
Telex: 25146 HEWPACK
Cable: HEWPACK Caracas
A,CP,E,MS,P

YUGOSLAVIA
Iskra-Commerce-Representation of Hewlett-Packard
Sava Centar Delegacija 30
Milentija Popovica 9
11170 ***BEOGRAD***
Tel: 638-762
Telex: 12042, 12322 YU SAV CEN

Iskra-Commerce-Representation of Hewlett-Packard
Koprska 46
61000 ***LJUBLJANA***
Tel: 321674, 315879
Telex:

EUROPEAN AREAS NOT LISTED, CONTACT
Hewlett-Packard S.A.
7 Rue du Bois-du-Lan
CH-1217 **MEYRIN 2**, Switzerland
Tel: (022) 83-81-11
Telex: 27835 hpse
Cable: HEWPACKSA Geneve

EAST EUROPEAN AREAS NOT LISTED, CONTACT
Hewlett-Packard Ges.m.b.h.
Wehlistrasse 29
P.O. Box 7
A-1205 **VIENNA**
Tel: (222) 35-16-210
Telex: 135823/135066

MEDITERRANEAN AND MIDDLE EAST AREAS NOT LISTED, CONTACT
Hewlett-Packard S.A.
Mediterranean & Middle East Operations
35, Kolokotroni Street
Platia kefallariou
GR-Kifissia, **ATHENS**, Greece
Tel: 808-0359, 808-0429
Telex: 21-6588
Cable: HEWPACKSA Athens

INTERNATIONAL AREAS NOT LISTED, CONTACT
Hewlett-Packard Co.
Intercontinental Headquarters
3495 Deer Creek Road
PALO ALTO, CA 94304
Tel: (415) 857-1501
Telex: 034-8300
Cable: HEWPACK

HEWLETT PACKARD

MANUAL PART NO. 92069-90001
Printed in U.S.A. July 1980

HEWLETT-PACKARD COMPANY
Data Systems Division
11000 Wolfe Road
Cupertino, California 95014